课程总目录

光澜跃迁　大模型算法与系统工程

	算法篇	推理系统篇
规模	17 章、六个部分、111 节、449 个知识点	26 章、七个部分、178 节、455 个知识点
动手内容	58 个动手任务、16 个章末产出	59 个实战项目，51 个可复现
硬件	52 个动手任务按 CPU 或 1x4090（≥24G）设计；4 个章末产出另给 1xH100 与 8xH100	必跑 10 个：2 个纯 CPU 可完成、6 个需 2x4090、2 个需 ncu 权限；其余 49 个建议跑或读透即可
随课交付	交付物 71 件；参考实现、配置与完整日志随该章文档交付	脚本 189、报告 130、结构化数据 681、运行日志 698、profile 产物 35
面试与就业	第 17 章整章：岗位能力图谱、面经题库、项目经历写法	295 道面试深挖题，随各章文档交付
边界	全篇 1 处只讲原理、不给实测数	12 个项目实测结论为不成立

算 法 篇

大模型算法进阶：训练、对齐与多模态

《大模型算法与系统工程》（训练、对齐与推理）**算法部分**

17 章教材、六个部分、111 节、449 个知识点，另有 16 个章末产出

章末产出由学员在标注硬件上完成，参考实现、配置与完整日志随该章文档交付

内容形态

从零实现 不调库、自己写出来　**实操** 用框架跑通并调出结果

源码精读 逐行读主流开源框架的实现

原理演示 单机环境不具备跨节点条件，只讲原理、不给实测数（全篇 1 处：5.5.4 跨节点 IB / RoCE 与机内 NVLink 的带宽量级差）

项目性质

生产级项目 只指用工业界真实在用的框架与工程标准做

可复现 学员自己能跑通，可写进简历

案例解读（已脱敏） 拆解工业界的真实做法与 trade-off，重在读懂深度与细节，不以复现为目标

硬件门槛

基础档（CPU）/ 基础档（1x4090） 按 CPU 或 1x4090（≥24G）设计，各项目所需显存标注在该章文档

进阶档（1xH100） 80G，可选进阶

观摩（8xH100） 交付完整配置与日志；可按“基于完整训练记录的理解与复盘”写进简历

一行标 2–3 个硬件档＝任选一档，差别在模型规模

1 大模型基础与 Transformer 原理

Self-Attention、RoPE 与 KV Cache；KV Cache 省下的是计算，换来的是显存与 I/O，长序列下主导复杂度的是序列长度的二次项，不是隐藏层维度

1.1 Tokenizer 与 Embedding

- 1.1.1 分词器（Tokenizer）与 token_ID
- 1.1.2 词嵌入层（Embedding）的查表机制

1.2 位置编码：PPL 平坦不等于外推成功

- 1.2.1 绝对位置编码（sin / cos）
- 1.2.2 相对位置编码与 RoPE 的旋转矩阵推导
- 1.2.3 RoPE 为什么融入 Q・K 计算而非独立于注意力
- 1.2.4 正弦余弦位置编码 **从零实现** 基础档（CPU）
- 1.2.5 上下文外推之一：位置插值 PI 与 NTK-aware 的缩放思路
- 1.2.6 上下文外推之二：YaRN 的分频缩放与注意力温度，以及哪些模型走它（DeepSeek-V3、gpt-oss），哪些不走（Llama 3.1 系用自家的分段波长缩放）
- 1.2.7 自训一个小模型（训练窗口 256），改 RoPE base 与缩放系数后评到 1–2k：画 PPL 随长度的曲线，并验证“PPL 平坦 ≠ 外推成功” **从零实现** 基础档（CPU）

1.3 自注意力机制（Self-Attention）

- 1.3.1 Query / Key / Value 的含义
- 1.3.2 缩放点积注意力（Scaled Dot-Product Attention）
- 1.3.3 Attention Mask 与 Causal Mask
- 1.3.4 Self-Attention 与 mask **从零实现** 基础档（CPU）

1.4 多头注意力（Multi-Head Attention）

- 1.4.1 多头子空间与拼接整合
- 1.4.2 Multi-Head Attention **从零实现** 基础档（CPU）

1.5 前馈网络、残差连接与层归一化：Post-LN 与 Pre-LN 的差别在梯度流

- 1.5.1 Layer Norm 与 Batch Norm 对比
- 1.5.2 NLP 为什么用 LN
- 1.5.3 Post-LN 与 Pre-LN 的梯度流对照，以及 RMSNorm 为什么取代 LayerNorm **从零实现** 基础档（CPU）

1.6 Encoder / Decoder 的结构与差异：双向还是因果，由 mask 矩阵决定

- 1.6.1 mask 矩阵决定的双向与因果可见性

1.6.2 为什么主流生成式大模型走 Decoder-only
1.6.3 Encoder-Decoder 顶层结构 **从零实现** 基础档（CPU）

1.7 **计算复杂度推导：长序列下由序列长度的二次项主导，不是隐藏层维度**
1.7.1 矩阵乘法与 Self-Attention / FFN 的计算量
1.7.2 序列长度 n 与隐藏层维度 d 的二次项主导

1.8 **KV Cache 与 Prefill / Decode：省下的是计算，换来的是显存与 I/O**
1.8.1 训练并行、推理串行的本质
1.8.2 KV Cache 的复杂度收益与显存、I/O 代价
1.8.3 Prefill 计算密集与 Decode 逐 token 生成

1.9 **章末产出：从零实现 Decoder-only** 基础档（CPU） **章末产出**
1.9.1 把 1.2–1.6 的组件拼成完整模型
1.9.2 与 HuggingFace 实现逐层对拍，报最大误差
1.9.3 **交付物：**对拍脚本、逐层误差表、逐层对拍的误差归因

2 主流架构精读与演进

注意力变体、MHA/GQA/MLA 与 MoE；MLA 要做低秩压缩就必须把 RoPE 解耦出来，MoE 不加负载均衡则路由会坍塌到少数专家

2.1 **注意力模式与掩码设计**
2.1.1 因果注意力与双向注意力
2.1.2 滑动窗口 / 局部 / 全局 / 扩张注意力
2.1.3 随机注意力、Longformer 与 BigBird
2.1.4 九种注意力掩码生成与对比 **从零实现** 基础档（CPU）

2.2 **稀疏注意力的演进：DeepSeek 主线走的是 DSA，NSA 未进入线上版本（截至 2026 年 9 月）**
2.2.1 提出背景与两大优势
2.2.2 压缩 / 选择 / 滑窗（sliding window）三分支的计算流程
2.2.3 计算复杂度分析
2.2.4 工程实现的潜在劣势
2.2.5 DSA 与 NSA 的差异：DeepSeek 为什么在 V3.2 用 DSA 上主线：截至 2026-08，其已发布模型走的是 DSA 与 V4 的 CSA+HCA，未见 NSA 上线，两者在选择粒度与训练成本上的 trade-off
2.2.6 DSA 之后：DeepSeek-V4 的 CSA 与 HCA 混合，把 KV 压缩推到了哪一步
2.2.7 NSA 三分支注意力 **从零实现** 基础档（CPU）

2.3 **注意力头与 KV 结构演进：MLA 要低秩压缩，就必须把 RoPE 解耦出来**
2.3.1 MHA 与 KV Cache 的显存压力

2.3.2 GQA / MQA 的分组共享与 KV Cache 显存收益
2.3.3 MLA 低秩压缩与 RoPE 解耦
2.3.4 DeepSeek-V3 Attention 逐行精读 **源码精读** 基础档（CPU）

2.4 **MoE 架构：不加负载均衡，路由会坍塌到少数专家**

2.4.1 Top-k 路由、共享专家与路由专家
2.4.2 moe_infer 的数据重排
2.4.3 负载均衡：aux loss、分组路由与 DeepSeek-V3 的 aux-loss-free 偏置法
2.4.4 MoE 与 MLP 的计算量对比及通信成本
2.4.5 MoE 门控路由与专家分发 **从零实现** 基础档（1x4090）

2.5 **DeepSeek-V4 架构精读**

2.5.1 V4-Pro 1.6T 总参 / 49B 激活 与 V4-Flash 284B / 13B：两档激活比对照，各自在用什么换什么
2.5.2 mHC：把残差映射约束到双随机矩阵流形
2.5.3 Muon 优化器在这套架构上的位置（训练侧展开见 5.1.6）
2.5.4 与 gpt-oss 的对照：Attention Sink、隔层滑窗、top-4 路由（softmax 在选中专家上做），哪些还成立、哪些被换掉了
2.5.5 源码核心解读

2.6 **非 Transformer 与混合架构：线性只换一部分层，全换会在特定任务上掉点**

2.6.1 线性注意力：把 softmax 换成核函数，复杂度从 $O(n^2)$ 降到 $O(n)$
2.6.2 状态空间模型与 Mamba：选择性状态与硬件感知的扫描实现
2.6.3 GDN 与混合搭法：Qwen3-Next 为什么只把一部分层换成线性
2.6.4 代价在哪：什么任务上会掉点，为什么长上下文是它的主场

2.7 **章末产出：KV 结构对照与 MoE 路由坍塌复现** 基础档（1x4090） **章末产出**

2.7.1 MHA / GQA / MQA / MLA 四种结构的显存曲线与质量对照
2.7.2 关掉负载均衡损失，复现路由坍塌并可视化路由分布
2.7.3 **交付物：**显存—质量对照表、路由分布图、结构选型结论与坍塌成因

第二部分 训练与微调

3 SFT 原理、数据构造与质量评价

信息度量、Loss Mask 与指令数据；attention_mask 不是 loss mask，混用会把 prompt 也算进损失，而 PPL 低同样不等于数据好

3.1 **模型的基本文件结构**

3.1.1 config.json 关键字段（GQA / RoPE / sliding_window）

3.1.2 chat_template 与 generation_config
3.1.3 safetensors 权重分片的原因

3.2 **语言模型的信息度量：最小化交叉熵等价于最小化前向 KL**

3.2.1 自信息、熵、交叉熵与 KL 散度
3.2.2 前向 KL 与反向 KL
3.2.3 最小化交叉熵为何等价于最小化 KL

3.3 **SFT 的原理：attention_mask 不是 loss mask，混用会把 prompt 也算进损失**

3.3.1 Next-Token Cross Entropy 与极大似然
3.3.2 预训练与 SFT 的 loss 差异
3.3.3 Assistant-only Loss 与 ignore_index(-100)
3.3.4 attention_mask 与 loss mask 的区别
3.3.5 ChatML 多轮对话拼接
3.3.6 ignore_index 与 loss mask 对照实验 **从零实现** 基础档（CPU）

3.4 **指令数据构造**

3.4.1 ChatML / Alpaca / ShareGPT 三种主流格式
3.4.2 冷启动数据工程：种子数据 + 场景库 + 特征库
3.4.3 并发调用 API 批量生成指令数据 **实操** 基础档（CPU）

3.5 **数据质量评价：PPL 低不等于数据好，还要看多样性与重复率**

3.5.1 困惑度（Perplexity）测试
3.5.2 多样性与重复度测试
3.5.3 大模型评价与主观评价
3.5.4 PPL 与 n-gram 重复率评测 **从零实现** 基础档（1x4090）

3.6 **章末产出：loss mask 对照实验与数据合成流水线** 基础档（1x4090） **章末产出**

3.6.1 ignore_index 设错与设对，两条 loss 曲线的差异与归因
3.6.2 从种子数据到质量筛选的完整合成流水线
3.6.3 **交付物：**双 loss 曲线、数据集、质量报告、loss mask 与数据质量的影响对照

4 高效微调方法谱系

LoRA、QLoRA 与 LLaMA-Factory 全流程；秩不是越大越好，显存与质量之间是一条帕累托曲线，本章把它实测画出来

4.1 **高效微调方法谱系：秩不是越大越好，显存与质量是一条帕累托曲线**

4.1.1 LoRA：低秩假设、秩的选择与显存收益
4.1.2 多适配器的模块化与推理时切换
4.1.3 Adapter 与 AdapterFusion

4.1.4 Prefix Tuning 的重参数化与实验结论

4.1.5 QLoRA 4-bit 加载与适配器训练 **实操** 基础档（1x4090）

4.2 **用 transformers 手写训练代码（单机单卡）**

4.2.1 数据预处理与 label 掩码

4.2.2 Trainer + DeepSpeed ZeRO-2

4.2.3 手写 train.py 与启动脚本 **从零实现** 基础档（1x4090）

4.3 **LLaMA-Factory 全流程**

4.3.1 数据集注册与 yaml 配置

4.3.2 LoRA 开关与 DeepSpeed 切换

4.3.3 SFT / DPO / KTO 三种数据格式的差异

4.4 **章末产出：LoRA 秩与显存—质量帕累托曲线** 基础档（1x4090） 进阶档（1xH100） **章末产出**

4.4.1 1x4090 档：1.5B 模型跑 r=4/8/16/32/64 五档完整流程

4.4.2 1xH100 档：7B 同实验

4.4.3 **交付物：**数据构造说明、帕累托图、选秩决策表、显存实测、精度对拍、留出集上的最终指标、秩选择的 trade-off 理由

4.4.4 两档怎么选：1x4090 与 1xH100 各自能做到什么、卡在哪

4.4.5 复现边界：哪些结论能复现；哪些数字复现不出来、差多少量级、为什么；差异说明了什么

第三部分 分布式训练与预训练

5 分布式训练

训练特有的显存占用、带反向的切分与并行选型；MLP 先列后行不是工程习惯，是 GeLU 不与加法交换的结果，而单机 8 卡量出来的结论不能外推到多机

5.1 **训练特有的显存占用：重计算用时间换显存，长序列下收益有边界**

5.1.1 训练显存的五大来源与混合精度下的显存分布

5.1.2 activation checkpointing：用时间换显存

5.1.3 FlashAttention backward：不存注意力矩阵、重算换显存

5.1.4 重计算在长序列下的收益边界

5.1.5 优化器状态从哪来：AdamW 的一阶、二阶动量与它在显存里的占比

5.1.6 Muon：矩阵结构的更新规则、它为什么能稳住大规模训练，以及对优化器状态显存的影响

5.1.7 重计算的显存—吞吐权衡实测 **实操** 基础档（1x4090）

5.2 ZeRO：Stage 1／2／3 依次切分优化器状态、梯度与参数，分片范围越大通信量越大

- 5.2.1 通信原语 All-Reduce / All-Gather / ReduceScatter 的语义与通信量
- 5.2.2 ZeRO 1 / 2 / 3 各自切分的对象：优化器状态、梯度、参数的显存占比
- 5.2.3 通信量分析：Stage 1/2 与 DP 持平、Stage 3 约 1.5 倍
- 5.2.4 ZeRO-Offload 与 DeepSpeed 参数解析
- 5.2.5 ZeRO 1 / 2 / 3 切分的 CPU 实现与显存占用对拍 **从零实现** 基础档（CPU）
- 5.2.6 ZeRO-Offload 开关的显存与吞吐实测：Stage 1/2/3 的分片收益在单卡上为什么看不见 **实操** 基础档（1x4090）

5.3 张量并行与流水线并行：MLP 先列后行不是习惯，是 GeLU 不与加法交换

- 5.3.1 行并行与列并行：前向切分与反向梯度聚合是一对共轭算子
- 5.3.2 MLP 为什么先列后行：GeLU 非线性不与加法交换，行切会在它之前逼出一个同步点，列切正是把这个同步点省掉
- 5.3.3 张量并行的前向切分与反向梯度聚合：双向数值等价验证 **从零实现** 基础档（CPU）
- 5.3.4 气泡问题与 micro-batch：1F1B 的意义就在反向重排
- 5.3.5 GPipe / 1F1B / Interleaved 1F1B 的调度时序对照
- 5.3.6 1F1B 调度模拟：含反向段与在飞激活曲线 **从零实现** 基础档（CPU）

5.4 上下文并行与专家并行：分段 softmax 的反向也能分段做，这是 Ring Attention 成立的前提

- 5.4.1 Ulysses 与 Ring Attention：两种切法，以及它们各自的反向代价
- 5.4.2 分段 Softmax 的前向与反向：为什么反向也能分段做
- 5.4.3 Ring Attention 分段 softmax 的前向与反向数值等价验证 **从零实现** 基础档（CPU）
- 5.4.4 专家并行的 aux loss：它怎么加进 loss、又怎么回传到路由
- 5.4.5 All-to-All 与 EP 路由语义的 CPU 实现，含 aux loss 的反向 **从零实现** 基础档（CPU）

5.5 怎么选与边界：单机 8 卡量出来的结论不能外推到多机

- 5.5.1 5D 并行（DP / TP / PP / CP / EP）的组合怎么定：先按哪一维卡显存来切
- 5.5.2 TP>1 时开 EP 要不要一并开 SP：Megatron-Core 版本决定它是硬约束还是建议
- 5.5.3 显存恒定律与百万长文本：上下文并行把什么变成了常数
- 5.5.4 跨节点 IB / RoCE 与机内 NVLink 的带宽量级差 **原理演示**
- 5.5.5 本课程哪些结论止步于单机：判断边界在哪，以及为什么单机 8 卡的多机结论不能直接外推

5.6 章末产出：四个反向的等价验证、气泡率与显存权衡 基础档（CPU） 基础档（1x4090） **章末产出**

- 5.6.1 四个从零实现的双向数值等价验证：张量并行梯度聚合、1F1B 反向段、Ring Attention 反向、EP aux loss 回传
- 5.6.2 不同 micro-batch 数下的气泡率模拟与理论对照
- 5.6.3 开关重计算、切换 ZeRO Stage 的显存峰值与吞吐对照

5.6.4 **交付物：**四合一双向等价验证报告、气泡率曲线、显存权衡曲线、四个反向的误差来源与并行选型结论

6 预训练与大规模训练实战

数据工程、Megatron 全流程与 MoE 的 FP8 训练；计入推理成本后 Chinchilla 最优点会往“更小、训更久”偏，而 Megatron 的并行配置项互相约束，embedding 切分是最常踩的一处

6.1 预训练数据工程

6.1.1 去重算法：编辑距离 / MinHash-LSH / SimHash / 语义嵌入

6.1.2 质量分类：FastText / PPL / BERT 三级

6.1.3 关键词掩盖与假数据脱敏

6.1.4 大规模数据去重流水线 **从零实现** 基础档（CPU）

6.2 Scaling Laws 与预训练配方：计入推理成本后，Chinchilla 最优点往“更小、训更久”偏

6.2.1 Chinchilla 计算最优：参数量与 token 量怎么配，以及它只对训练算力最优，计入推理成本后最优点往“更小、训更久”偏

6.2.2 数据配比与退火：多语料混合比例、后期高质量退火

6.2.3 学习率调度：warmup、cosine 与 WSD 的 trade-off

6.2.4 训练前怎么估：给定卡时预算，反推模型规模与数据量

6.3 Megatron / Mcore 的并行配置与训练全流程：配置项互相约束，embedding 切分是最常踩的一处

6.3.1 Mcore 的并行配置项与相互约束

6.3.2 HF 与 Mcore 的格式互转（Megatron-Bridge）

6.3.3 embedding 切分陷阱

6.3.4 数据处理：jsonl → bin / idx

6.3.5 训练启动、checkpoint 保存与断点恢复

6.3.6 单机 8 卡端到端训练 **实操** 观摩（8xH100）

6.4 大规模 MoE 的 FP8 训练

6.4.1 FP8 格式选择：E4M3 / E5M2 / hybrid

6.4.2 缩放策略、amax 历史窗口与精度兜底

6.4.3 MoE 路由器与令牌调度参数

6.4.4 MoE 的 FP8 训练与精度对照 **实操** 观摩（8xH100）

6.5 DeepSeek-V3 的 FP8 精度方案：算在 Tensor Core，累加放回 CUDA Core

6.5.1 激活 1×128 与权重 128×128 细粒度量化

6.5.2 三个 FP8 GEMM 的张量统一 E4M3；MoE 通信 dispatch 走 FP8、combine 走 BF16

6.5.3 Tensor Core 计算与 CUDA Core 高精度累加

6.6 **章末产出：Megatron 端到端训练与 FP8 精度对照** 观摩（8xH100） **章末产出**

6.6.1 从数据准备到 checkpoint 恢复的完整链路跑通

6.6.2 BF16 与 FP8 两条训练曲线的对照与精度损失量化

6.6.3 **交付物：**BF16/FP8 双 loss 曲线、吞吐与 MFU 实测、全流程日志、配置说明、精度对拍、FP8 掉点的归因

第四部分 对齐、推理能力与评测

7 强化学习基础

MDP、Actor-Critic 与语言模型的对应；把生成看成决策之后，整条轨迹只有一个奖励，语言模型的 RL 从一开始就是稀疏奖励问题

7.1 **强化学习核心逻辑**

7.1.1 MDP 与贝尔曼方程

7.1.2 Q 与 V 函数、探索与利用

7.1.3 动态规划 / 蒙特卡罗 / 时序差分

7.1.4 Actor-Critic 方法

7.1.5 玩具环境上的 Actor-Critic **从零实现** 基础档（CPU）

7.2 **从 RL 到语言模型：整条轨迹只有一个奖励，这是稀疏奖励问题**

7.2.1 状态、动作、轨迹在生成任务里对应什么

7.2.2 为什么语言模型的 RL 是稀疏奖励问题

7.2.3 与经典 RL 的三点关键差异

7.3 **章末产出：Actor-Critic 收敛与超参敏感性** 基础档（CPU） **章末产出**

7.3.1 玩具环境跑通并画出收敛曲线

7.3.2 学习率、折扣因子、熵系数的敏感性扫描

7.3.3 **交付物：**收敛曲线、敏感性表、超参敏感性结论与失败模式

8 RLHF 与偏好优化

PPO、Reward Modeling 与 DPO；PPO 的 Clipping 抑制的是重要性采样带来的方差，DPO 把奖励模型重参数化成隐式奖励，代价写在它的适用边界上

8.1 **RLHF 与 PPO：Clipping 抑制的是重要性采样带来的方差**

8.1.1 Actor / Critic / Reward / Reference 四模型组成

8.1.2 Rollout 与 Update 双阶段流程

8.1.3 PPO-Clip 与 PPO-惩罚两种目标形式

8.1.4 重要性采样引入的方差与 Clipping 的抑制

8.1.5 Reward Modeling 与 pairwise rank loss

8.1.6 GAE 优势函数

8.1.7 per-token KL 估计

8.1.8 PPO 四模型完整训练循环 **从零实现** 基础档（1x4090）

8.2 **DPO（Reward-Model-Free）：奖励模型被重参数化成隐式奖励，代价写在适用边界上**

8.2.1 重参数化与隐式奖励

8.2.2 Bradley-Terry 损失与梯度机制

8.2.3 两阶段 DPO 实战（拒答与安全对齐）

8.2.4 DPO 与 PPO 的适用边界

8.2.5 DPO 损失函数与偏好准确率监控 **从零实现** 基础档（1x4090）

8.3 **章末产出：Reward Model 与 DPO 双路线对照** 基础档（1x4090） **章末产出**

8.3.1 训练 Reward Model 并监控 pairwise 准确率

8.3.2 同一份偏好数据分别走 PPO 与 DPO，对照结果与成本

8.3.3 **交付物：**两条训练曲线、成本对照表、两条路线的适用边界

9 可验证奖励与 RL 算法演进

RLVR、GRPO、DAPO、GSPO 与 VERL；GRPO 用组内相对优势替掉价值模型，而熵崩溃与熵爆炸这两类相反的失稳，调的是同一组参数

9.1 **RLVR 与 GRPO：组内相对优势替掉价值模型，PRM 则在训练时用不起来**

9.1.1 可验证奖励的适用场景

9.1.2 组内相对优势替代价值模型

9.1.3 结果监督与过程监督的 token 级映射

9.1.4 PRM 为什么在训练时用不起来：步骤难定义、中间步难判、必然 reward hacking

9.1.5 GRPO 组内相对优势计算与数值验证 **从零实现** 基础档（CPU）

9.2 **DAPO**

9.2.1 Clip-Higher 非对称截断

9.2.2 动态采样

9.2.3 Token 级梯度加权

9.2.4 超长惩罚与移除显式 KL

9.3 **GSPO：序列级比率换来的是对训练—推理精度差异的容忍**

9.3.1 序列级重要性比率

9.3.2 MoE 训练稳定性与免 Routing Replay

9.3.3 对训练—推理引擎精度差异的容忍

9.4 **熵崩溃与熵爆炸：两类失稳现象相反，调的却是同一组参数**

9.4.1 两类失稳的现象识别

9.4.2 KL 系数、熵系数与学习率的调整

9.4.3 采样温度与 Reward 归一化

9.4.4 熵崩溃的复现与调参恢复 **实操** 进阶档（1xH100）

9.5 **VERL 框架与源码精读**

9.5.1 Rollout 执行与 DataProto 结构

9.5.2 Reward Manager 与 compute_score 路由

9.5.3 Advantage 估计与广播机制（GAE / GRPO / RLOO）

9.5.4 Policy Loss 与聚合方式

9.5.5 训练步骤、反向传播与分布式梯度处理

9.5.6 训练稳定性技巧

9.5.7 VERL 多卡 GRPO 训练启动 **实操** 观摩（8xH100）

9.6 **章末产出：跑通一个完整的 RLVR 任务** 基础档（1x4090）进阶档（1xH100）观摩（8xH100） **章末产出**

9.6.1 1x4090 档：0.5B 玩具任务，跑通采样—打分—更新完整循环

9.6.2 1xH100 档：1.5B 数学推理任务；8xH100 档：7B + 共置与分离两种架构对照

9.6.3 奖励设计说明：为什么这么设、试错过的哪几版不行

9.6.4 **交付物：**prompt 集构造与难度分层、训练曲线、奖励设计文档、稳定性排查记录、三条基线对照、训练不稳的归因与处置

9.6.5 三档怎么选：1x4090 / 1xH100 / 8xH100 各自能做到什么、卡在哪

9.6.6 复现边界：哪些结论能复现；哪些数字复现不出来、差多少量级、为什么；差异说明了什么

10 模型蒸馏

在线策略蒸馏、离线蒸馏与采样框架的复用；离线蒸馏的问题在于 teacher 的分布不是 student 会走到的分布，而长尾退化是蒸馏的结构性代价

10.1 **离线蒸馏：teacher 的分布不是 student 会走到的分布**

10.1.1 logits 蒸馏与温度的作用

10.1.2 中间层蒸馏与特征对齐

10.1.3 黑盒数据蒸馏：只有输出文本时的构造方法

10.1.4 离线蒸馏的训练—推理分布不匹配

10.1.5 logits 蒸馏与温度扫描 **从零实现** 基础档（1x4090）

10.2 **在线策略蒸馏（OPD）：与 GRPO 一样的采样，不一样的监督来源**

10.2.1 student 自己采样，teacher 在 student 的轨迹上逐 token 给分布

10.2.2 反向 KL 作为密集监督信号

10.2.3 与 GRPO 的同框架异信号：一样的采样，不一样的奖励来源

10.2.4 收敛速度与数据效率的对比

10.2.5 OPD 训练循环 **从零实现** 进阶档（1xH100）

10.3 **蒸馏的边界：长尾退化是蒸馏的结构性代价**

10.3.1 蒸馏与量化的 trade-off：什么时候该压参数，什么时候该压精度

10.3.2 teacher 能力上限与 student 容量的匹配

10.3.3 蒸馏出来的模型为什么常在长尾上退化

10.4 **章末产出：离线蒸馏与 OPD 同任务对照** 进阶档（1xH100） **章末产出**

10.4.1 同一个 teacher、同一个 student、同一份任务，两条路线各跑一遍

10.4.2 在 student 自己会走到的分布上评测，看差距出在哪

10.4.3 **交付物：**质量—成本对照表、分布差异分析、两种蒸馏的适用条件

11 长思维链系统

RL 训练与推理服务的协同设计；长 CoT 上线后指标会重排，TTFT 让位于 TPOT × 输出长度，而 test-time scaling 有边界，加算力不会一直涨

11.1 **RL 训练的成本结构：优化采样吞吐就是直接降训练成本**

11.1.1 rollout 与 update 的墙钟时间占比

11.1.2 采样组规模 G 与长 CoT 的 token 量级

11.1.3 为什么优化采样吞吐等于直接降低训练成本

11.2 **rollout 的系统优化：同前缀成组采样，使前缀缓存命中率接近 100%**

11.2.1 前缀缓存、continuous batching、投机解码（Speculative Decoding）与 PD 分离

11.2.2 前缀缓存在 GRPO 上命中率为什么接近 100%

11.2.3 一组采样的长度方差对批处理的冲击

11.2.4 rollout 的硬 deadline：在线服务没有的约束

11.2.5 投机解码为什么能无损加速 rollout

11.2.6 前缀缓存开关的 A/B 与吞吐对照 **实操** 基础档（1x4090）

11.3 **训练与推理的精度不一致：差异随序列长度累积，长 CoT 放大得最快**

11.3.1 两侧 log-prob 差异从哪来

11.3.2 差异如何混进重要性采样比

11.3.3 表现形态：方差异常、clipping 频发、熵崩溃

11.3.4 GSPO 的序列级比率为什么更耐受

11.3.5 不同 backend 的精度差异：怎么测出来、差多少个量级

11.3.6 差异随序列长度的累积：长 CoT 为什么放大得更快

11.3.7 工程上怎么对齐两侧：采样参数、kernel 选择与精度配置的一致性检查

11.3.8 log-prob 差异测量与 ratio 影响 **从零实现** 进阶档（1xH100）

11.4 **权重同步与架构选择**

11.4.1 同步代价与卡数配比的 trade-off 框架：落盘重载 / NCCL 广播 / 共享显存三种代价，共置分时复用与分离部署的选择，两侧卡数按时间占比推导

11.4.2 与推理侧 PD 分离的同构关系

11.5 **长 CoT 上线后的指标重排：TTFT 让位于 TPOT × 输出长度**

11.5.1 KV Cache 显存爆炸与逐退策略

11.5.2 TTFT 让位于 TPOT × 输出长度

11.5.3 长度不可预测对调度与准入的影响

11.5.4 思考预算控制：截断、early-exit 与 budget forcing

11.6 **推理时的算力扩展：test-time scaling 有边界，加算力不会一直涨**

11.6.1 Best-of-N 与 self-consistency

11.6.2 PRM 与 ORM：这里的 PRM 是推理时的打分器

11.6.3 PRM 引导的树搜索

11.6.4 test-time scaling law 与它的边界

11.7 **训练与推理的成本分配：搜索成本要么前置到训练，要么后置到推理**

11.7.1 RL 把搜索成本前置到训练

11.7.2 test-time search 把成本后置到推理

11.7.3 真实系统里两者怎么组合

11.8 **章末产出：训练框架与推理引擎的精度差异测量** 进阶档（1xH100） **章末产出**

11.8.1 同一份权重，两侧各算一遍 log-prob，量出差异分布

11.8.2 把差异注入 ratio，观察方差与 clipping 触发率的变化

11.8.3 **交付物：**差异分布图、ratio 方差对照、精度差异的可接受范围

12 模型能力评测

benchmark、数据污染与业务场景下的盲评；榜单成绩不能直接横向比较，“提升 2 个点”要先通过置信区间与最小可检测差异的检验才算数

12.1 **benchmark 的构成与局限：榜单成绩不能直接横向比较**

12.1.1 MMLU / CEVAL / CMMLU / GPQA 的题型与覆盖面

12.1.2 few-shot 设置与提示敏感性

12.1.3 榜单成绩为什么不能直接横向比较

12.2 **数据污染**

12.2.1 污染的三种来源

12.2.2 n-gram 重叠检测

12.2.3 困惑度异常检测
12.2.4 污染检测与成绩重估 **从零实现** 基础档（1x4090）

12.3 **LLM-as-judge：位置、长度与自我偏好三种偏差要先校准**
12.3.1 位置偏差、长度偏差与自我偏好
12.3.2 校准方法与人工对照
12.3.3 什么时候不能用模型当裁判

12.4 **统计口径：“提升 2 个点”要先通过置信区间与最小可检测差异的检验**
12.4.1 pairwise 胜率与置信区间
12.4.2 样本量与最小可检测差异
12.4.3 “提升 2 个点”是否有统计意义

12.5 **业务场景下的评测设计** **案例解读（已脱敏）**
12.5.1 没有现成 benchmark 时，怎么定义改进
12.5.2 AB 盲评的流程设计与常见陷阱
12.5.3 从人工评价为主，逐步过渡到模型评价为主
12.5.4 评价一致性的持续监控
12.5.5 AB 盲评设计与执行 **从零实现** 基础档（1x4090）

12.6 **评测驱动的迭代**
12.6.1 每次改动都要能给出改进的依据
12.6.2 评测集的版本管理与防过拟合

12.7 **章末产出：一份站得住的模型评测报告** 基础档（1x4090） **章末产出**
12.7.1 对第 4 章或第 8 章训出的模型做完整评测
12.7.2 污染检测 + 盲评 + 置信区间，三样齐全
12.7.3 **交付物：**各 benchmark 得分对照表（含置信区间）、评测报告、污染检测结果、盲评原始数据、结论的置信边界

第五部分 应用方向

13 RAG 检索增强生成

召回、重排与 RAFT；BM25 与语义召回的失败案例不重合，混合才有效，而传统 RAG 在多跳问题上会失效

13.1 **RAG 的适用条件**
13.1.1 知识局限与时效性
13.1.2 更新成本与黑箱可解释性
13.1.3 检索与生成解耦的模块化价值

13.1.4 什么时候不该用：长上下文直读、agentic search、纯精确匹配

13.2 **BERT 与双向注意力**

13.2.1 MLM 与 NSP 任务

13.2.2 WordPiece 与 BPE 的合并准则差异

13.2.3 三个 Embedding 层

13.2.4 BERT 文本分类 / 重排 / NER 训练 **实操** 基础档（1x4090）

13.3 **稠密召回**

13.3.1 Sentence-BERT 与对比学习

13.3.2 MSE / InfoNCE / Triplet 三种损失

13.3.3 温度系数、困难样本与 batch 规模

13.3.4 三种对比学习损失 **从零实现** 基础档（1x4090）

13.4 **稀疏召回：BM25 靠词频饱和与长度归一化压住长文档**

13.4.1 TF-IDF

13.4.2 BM25 的词频饱和与文档长度归一化（k1 / b）

13.4.3 TF-IDF 检索器 **从零实现** 基础档（CPU）

13.5 **混合与多路召回：BM25 与语义的失败案例不重合，混合才有效**

13.5.1 BM25 与语义召回的失败案例对照

13.5.2 分数归一化与加权混合排序

13.5.3 BM25 与语义的混合检索 **从零实现** 基础档（1x4090）

13.6 **分层召回**

13.6.1 分类召回方案

13.6.2 聚类召回（BGE + KMeans 两级检索）

13.6.3 聚类两级召回 **从零实现** 基础档（1x4090）

13.7 **向量库工程**

13.7.1 Faiss 索引选型：Flat / IVF_PQ / HNSW

13.7.2 GPU 加速与分片管理

13.7.3 nprobe 与检索深度调优

13.7.4 召回评价：召回率与 Top-N

13.7.5 Faiss 索引构建与检索 **实操** 基础档（1x4090）

13.8 **重排（Rerank）：双塔与交叉编码器之间是精度—延迟的 trade-off**

13.8.1 双塔与交叉编码器的精度—延迟 trade-off

13.8.2 Pairwise 与 Listwise 损失

13.8.3 混合负采样与查询改写

13.9 **RAFT 实战（法律问答）** **可复现**

13.9.1 三种训练路线的 trade-off

13.9.2 分类模型训练

13.9.3 对比学习样本构造（BM25 挖困难负样本）

13.9.4 基于召回文本的抗噪问答训练

13.9.5 分类模型与双编码器训练 **实操** 基础档（1x4090）

13.10 **Deep ReSearch：传统 RAG 在多跳问题上失效**

13.10.1 传统 RAG 的局限

13.10.2 实体锚定的知识增强与多跳问题合成

13.10.3 低阶与高阶 ReAct 数据构造

13.10.4 拒绝采样

13.11 **章末产出：端到端 RAG 系统**

基础档（1x4090）
进阶档（1xH100）
观摩（8xH100） **章末产出**

13.11.1 1x4090 档：混合召回 + 重排 + 生成的完整链路

13.11.2 1xH100 档：更大语料与更大重排模型；8xH100 档：分片索引与并行召回

13.11.3 **交付物：**评测集构造说明、召回率@K 曲线、端到端问答评测、消融表与分层 badcase 归因、召回与重排的 trade-off 记录

13.11.4 三档怎么选：1x4090 / 1xH100 / 8xH100 各自能做到什么、卡在哪

13.11.5 复现边界：哪些结论能复现；哪些数字复现不出来、差多少量级、为什么；差异说明了什么

14 Agent 与工具调用

训练、工程实现、评测与 AI-native 研发；Agent 框架有半衰期，选型判据比选哪个框架更耐用；评测评的是执行轨迹，不是 benchmark 分数

14.1 **什么是 Agent**

14.1.1 Agent 与对话模型的区别

14.1.2 LLM 核心、工具集、记忆、反馈、规划与执行六组件

14.2 **ReAct 格式与 tool_call 格式**

14.2.1 ReAct 轨迹：Thought / Action / Observation

14.2.2 Qwen3 的 tool_call 格式

14.2.3 并行工具调用与推理内容渲染

14.2.4 tools schema 注入与工具返回

14.2.5 chat_template 渲染与用例验证 **实操** 基础档（CPU）

14.3 **Agent 训练的 loss 设计：工具返回参与注意力但不计 loss**

14.3.1 工具返回参与注意力但不计 loss

14.3.2 ignore_index 在 Agent 场景的应用

14.4 Agent 训练数据构造

14.4.1 工具描述的字段设计

14.4.2 教师模型批量生成

14.4.3 简单与长程两类数据（长程的四步构造见 14.10）

14.4.4 plan 示例的人工迭代

14.5 ReAct 到 tool_call 的格式转换

14.5.1 中英文标签兼容的解析

14.5.2 非标准 JSON 的容错处理

14.5.3 ReAct → tool_call 批量转换脚本 **从零实现** 基础档（CPU）

14.6 Agent 框架与工作流编排：框架有半衰期，选型判据比选哪个框架更耐用

14.6.1 框架生态与选型判据：LangChain / LangGraph / Dify / AutoGen，框架的半衰期怎么判

14.6.2 LangGraph 核心机制：状态图、条件分支与循环、checkpoint 与断点续跑

14.6.3 自定义工具注册与接入自训模型：vLLM 的 OpenAI 兼容端点

14.6.4 用 LangGraph 把自训模型包成工具调用 Agent **实操** 基础档（1x4090）

14.7 开放域 Agent 的工程实现（以客服场景为例）

14.7.1 意图识别与请求路由

14.7.2 多轮对话的记忆管理：窗口截断 / 摘要压缩 / 检索式长期记忆

14.7.3 上下文工程：组装顺序、压缩策略与窗口预算分配

14.7.4 MCP 协议与工具接入

14.7.5 最小 MCP Server：协议握手、工具暴露与调用回环 **从零实现** 基础档（CPU）

14.7.6 多 Agent 协作：routing 设计、主从与对等模式、状态同步

14.7.7 monitor agent：在线表现的监控、Trace 采集与回流

14.7.8 工具调用失败的归因与治理：参数校验、重试降级、检索兜底（召回本身的优化见第 13 章）

14.7.9 多轮记忆管理器：三种记忆策略的同题对照 **从零实现** 基础档（CPU）

14.8 Agent 评测体系：评的是执行轨迹，不是 benchmark 分数

14.8.1 模型评测与 Agent 评测的分界：从 benchmark 到执行轨迹（模型侧评测见第 12 章）

14.8.2 双口径：工具调用准确率与端到端任务成功率、多轮一致性

14.8.3 evaluation harness：用例集设计、回归与 badcase 归因

14.8.4 最小 Agent 评测器：对执行轨迹计算双口径指标 **从零实现** 基础档（CPU）

14.9 Coding Agent 与 Harness：上下文管理、工具层与权限模型决定它能不能用，模型能力只是其中一项

14.9.1 Coding Agent 解剖：agent loop、上下文管理、工具层与权限模型（以 Claude Code / OpenClaw 为例）

14.9.2 Harness 与 Skills：SKILL.md 开放标准、自定义 Skill 与 Hook

14.9.3 为自训模型 Agent 写一个自定义 Skill 并经 MCP 接入工具 **实操** 基础档（1x4090）

14.9.4 沙箱、权限与 Agent 安全：prompt injection、工具滥用与隔离

14.9.5 把 AI Coding 用成稳定生产力：从零实现项目的 AI 辅助工作流与验收习惯

14.10 长程工具调用与 Agentic-RL 前沿

14.10.1 工具图谱构造

14.10.2 Plan 反向生成 Query

14.10.3 执行轨迹生成

14.10.4 数据质量与多样性把控

14.10.5 RLVR 的奖励设计

14.11 章末产出：客服 Agent 从数据构造到端到端评测

基础档（1x4090）
进阶档（1xH100）
观摩（8xH100） **章末产出**

14.11.1 1x4090 档：1.5B 模型 + 客服工具集，跑通构造—训练—评测

14.11.2 1xH100 档：7B + 长程工具链；8xH100 档：全参 Agentic-RL 训练（多轮工具调用 + RLVR 奖励）

14.11.3 **交付物：**数据集与构造说明、双口径成功率报告、三条基线对照、单次调用的成本与延迟、失败案例归因、治理优先级与改进项排序

14.11.4 三档怎么选：1x4090 / 1xH100 / 8xH100 各自能做到什么、卡在哪

14.11.5 复现边界：哪些结论能复现；哪些数字复现不出来、差多少量级、为什么；差异说明了什么

15 多模态基础

跨模态对齐、检索与数据工程；SigLIP 用 Sigmoid 换掉对称 InfoNCE，绕开 CLIP 的四点不足，而合成数据要先经过风险检测再进配比

15.1 跨模态检索：SigLIP 用 Sigmoid 换掉对称 InfoNCE，绕开 CLIP 的四点不足

15.1.1 对称 InfoNCE 与图文共享语义空间

15.1.2 零样本分类

15.1.3 CLIP 的四点不足

15.1.4 SigLIP 的 Sigmoid 改造思路

15.1.5 CLIP 图文对比学习核心 **从零实现** 基础档（1x4090）

15.2 跨模态细粒度搜索（自研 Col-MiniCPM-V） **可复现**

15.2.1 ColBERT 式 MaxSim 多向量匹配

15.2.2 切图保细节

15.2.3 与单塔 / 双塔的速度—精度对比

15.2.4 训练脚本与检索 demo **实操** 基础档（1x4090）

15.3 **多模态数据工程：合成数据要先经过风险检测再进配比**

15.3.1 图文对的清洗、去重与对齐度筛选

15.3.2 训练配比：纯文本、图文对、指令数据的比例怎么定

15.3.3 合成数据的风险与检测

15.3.4 数据配比消融实验 **实操** 基础档（1x4090）

15.4 **跨模态检索的评测**

15.4.1 召回率、MRR 与 nDCG

15.4.2 细粒度检索的困难样本构造

15.5 **章末产出：跨模态检索系统与零样本验证** 基础档（1x4090） **章末产出**

15.5.1 从零训练一个图文对比模型并做零样本分类

15.5.2 与开源基线在同一评测集上对照

15.5.3 **交付物：**训练曲线、零样本准确率、检索指标、检索失效的典型模式

16 多模态生成模型与训练

架构、高分辨率适配与视觉 Token 压缩；高分辨率的切法由评分函数选，不是固定切法，原生多模态也不是三段式的替代，部分场景下三段式仍更划算

16.1 **多模态生成模型的主流架构：三段式里 LLM 是中枢**

16.1.1 编码器 + 适配器 + LLM 三段式

16.1.2 为什么 LLM 是中枢

16.2 **多模态模型的六大问题**

16.2.1 细粒度丢失与幻觉

16.2.2 对齐税与训练推理成本

16.2.3 慢思考构建与位置编码对齐

16.3 **高分辨率适配（LLaVA-UHD）：切法由评分函数选，不是固定切法**

16.3.1 缩放 / Padding / 重叠切分 / 子图切分的对比

16.3.2 评分函数选最优切分

16.3.3 自适应切图 **从零实现** 基础档（1x4090）

16.4 **MiniCPM-V 系列** **可复现**

16.4.1 自适应切图流程

16.4.2 Resampler 压缩视觉 token

16.4.3 3D-Resampler 的视频时序分组

16.4.4 MiniCPM-O 的流式分块编解码

16.4.5 Resampler 与 3D-Resampler **从零实现** 基础档（1x4090）

16.5 Qwen3-VL 精读与微调

16.5.1 SigLIP-2 视觉编码器与动态分辨率：2D-RoPE 加绝对位置编码插值怎么吃下任意尺寸

16.5.2 Interleaved-MRoPE：t / h / w 三轴频率交错分配，与分块式 MRoPE 差在哪

16.5.3 DeepStack：三层 ViT 特征以残差方式注入 LLM 前三层

16.5.4 从 T-RoPE 到显式文本时间戳：视频时序定位怎么变准

16.5.5 四阶段预训练与三阶段后训练：上下文怎么一步步撑到 256K

16.5.6 LLaMA-Factory 微调多模态模型 **实操** 基础档（1x4090）

16.6 开源模型谱系

16.6.1 InternVL 的渐进式扩展策略

16.6.2 SmolVLM 的小模型工程经验

16.7 多模态推理模型

16.7.1 Skywork R1V 的推理能力迁移

16.7.2 R1V2 的 MPO + GRPO 混合与 SSB 机制

16.7.3 InternVL3 的 MPO

16.7.4 VLM 的 GRPO 微调：3B 模型 + LoRA + bf16 在可验证几何任务上的强化学习（全参 3B 在 24G 放不下） **实操** 基础档（1x4090）

16.8 原生多模态：统一架构不是三段式的替代，部分场景下三段式仍更划算

16.8.1 三段式的边界在哪：编码器 + 适配器 + LLM 各自丢失了什么信息

16.8.2 统一理解与生成的三条路线：纯自回归、纯扩散、自回归与扩散融合

16.8.3 视觉 token 化的四种做法：像素重建式、语义对齐式、可学习 query 式、混合式，各自把什么信息保住了

16.8.4 统一模型难在哪：tokenizer 选型、跨模态注意力设计、训练数据构造、评测缺位

16.8.5 什么场景下三段式仍然更划算

16.9 章末产出：多模态模型微调与切图策略对照

基础档（1x4090）
进阶档（1xH100）
观摩（8xH100） **章末产出**

16.9.1 1x4090 档：2B / 4B 模型 LoRA 微调，跑通完整链路（Qwen3-VL dense 无 3B 档）

16.9.2 1xH100 档：7B LoRA + 三种切图策略对照；8xH100 档：7B 全参

16.9.3 **交付物：**自建场景图文集与构造说明、微调模型、切图策略对照表、幻觉评测、badcase 归因链、切图策略的选择理由

16.9.4 三档怎么选：1x4090 / 1xH100 / 8xH100 各自能做到什么、卡在哪

16.9.5 复现边界：哪些结论能复现；哪些数字复现不出来、差多少量级、为什么；差异说明了什么

17 业务选型、项目经历与面试就业

选型判据、岗位能力图谱与面经题库；真实业务里决定选型的不是模型排名，是成本、延迟与可维护性这三条约束的排序

17.1 **真实业务中的模型选型：成本、延迟与可维护性三条约束的排序决定选型**

- 17.1.1 什么时候用纯 Prompt
- 17.1.2 什么时候用 RAG
- 17.1.3 什么时候 SFT / 对齐 / Agent 化
- 17.1.4 成本、延迟与可维护性三条约束怎么排序

17.2 **五个旗舰项目逐个精讲**

- 17.2.1 训练 / 微调项目（第 4 章）
- 17.2.2 对齐 / 强化学习项目（第 9 章）
- 17.2.3 RAG 项目（第 13 章）
- 17.2.4 Agent 项目（第 14 章）
- 17.2.5 多模态项目（第 16 章）

17.3 **岗位能力图谱**

- 17.3.1 大模型算法岗（训练 / 微调）能力要求
- 17.3.2 大模型推理 / Infra 岗能力要求
- 17.3.3 岗位 JD 的高频技能对照

17.4 **面试高频问题与答题思路：同一个问题的三层答法：跑通 → 讲原理 → 讲 trade-off**

- 17.4.1 架构类问题
- 17.4.2 训练、微调与对齐类问题
- 17.4.3 RAG / Agent / 多模态类问题
- 17.4.4 同一个问题的三层答法：跑通 → 讲原理 → 讲 trade-off

17.5 **大模型面经题库**

- 17.5.1 校招高频真题
- 17.5.2 社招与专家岗真题
- 17.5.3 工程排障题：loss 尖峰、数值精度与数据质量

17.6 **从学习到项目到就业**

- 17.6.1 项目经历的书面表达：这样写与别这样写
- 17.6.2 学完之后怎么继续：跟进哪些源码与论文、按什么顺序、怎么判断一个新方法值不值得学

大模型推理系统工程

《大模型算法与系统工程》（训练、对齐与推理）**推理系统部分**

26 章教材、七个部分、178 节、455 个知识点，另有 59 份实战项目文档

随课交付脚本 189 份、报告 130 份；另附运行日志 698、结构化数据 681、profile 产物 35

内容形态	**从零实现** 不调库，自己写出来（9 项）　**实操** 用框架跑通并调出结果（46 项） **源码精读** 逐行读主流开源框架的实现（1 项） **方案设计** 交付可直接送审的方案与配置文件，不含实测数据（3 项）
动手要求	**必跑** 核心收益来自动手，只读不能替代（10 项） **建议跑** 动手更好，读透日志、中间产物与结果也能掌握项目深度和细节（31 项） **读透即可** 动手成本高，收益主要在读懂设计与结果（18 项）
必跑的门槛	**纯 CPU** 没有 GPU 也能完整做完 **2x4090（≥24G）** 多数要卡的项目按这个配置设计，下限是 1x4090（≥24G）；实测在 1xH100（80G）上完成，4090 上的显存与耗时需复测，各项目所需显存标注在该章文档 **需 ncu 权限** 需要能开启 GPU 性能计数器的环境，多数云主机要单独申请，材料内附检查方法与替代路径
项目性质	默认**可复现**：学员自己能跑通，可写进简历（51 项）。 **案例解读** 拆解工业界的真实做法与取舍，重在深度、细节并读懂，不以复现为目标（8 项，标出）
实测环境	真机压测在 8 卡 NVIDIA H100 80GB HBM3 节点上做，按各章被测对象取 1 / 2 / 4 / 8 张，不是本课默认的学员配置；纯 CPU 项目不在其内。
版本口径	实测所用引擎版本早于课程锁定版本（vLLM 0.26.0 / SGLang 0.5.16），按锁定版复现时绝对值会有出入，趋势与结论不受影响。
实跑规模	多数项目的实跑量小于课程脚本的设计规模，绝对值只看趋势、不作通用基准；每项的实跑量与设计规模写在该项目下方。
示例输出	课程材料中用于演示脚本读法的示例输出为构造数据，已逐处标明。
结论方向	59 个项目中，12 个的实测结论是该优化或改动不成立。
章末产出	各章末一行按脚本、报告、结构化数据、日志、profile 产物五类计数；产物随各章文档交付。
代码	该小节配有可运行代码，随该章文档交付。
面试深挖题	每个项目另附 5 道，共 295 道；题目与参考答法随该章文档交付，不在目录内。

1　推理系统概述

延迟与吞吐指标的定义、分层架构与三种部署形态的取舍；kernel 时长榜上 GEMM 占 86% 不等于瓶颈在算子，也可能是批次未填满。

1.1　**推理系统的关键性能指标**

1.1.1　延迟指标

1.1.2　吞吐指标

1.1.3　指标间的制约关系

1.1.4　推理性能指标计算器与延迟-吞吐权衡分析　代码

1.2　**推理的完整生命周期**

1.3　**推理系统的分层架构**

1.4　**推理部署形态**

1.4.1　在线服务（Online Serving）

1.4.2　离线批处理（Offline Batch）

1.4.3　边缘部署（Edge Deployment）

1.4.4　端云协同（Edge-Cloud Hybrid）

1.5　**推理优化技术总览**

1.6　**主流推理框架对比**

章末产出　脚本 5　报告 3　结构化数据 21　日志 23　profile 产物 2

项目 1.1　**压测平台**　**实操** 建议跑

基于 vllm bench serve 搭建统一口径的企业压测平台，按多租户画像分段测出吞吐、TTFT、TPOT 基线，再用 kernel 时长榜把瓶颈定位到GEMM 占比高但批次未填满

长文档段 TTFT P99 1127 ms、ITL P99 190 ms；kernel 时长榜 GEMM 占 86%，瓶颈不在算子，在批次未填满。

单步 token 预算翻倍的复测未覆盖，本项目不给加速比结论。

项目 1.2　**多框架推理性能对比报告**　**实操** 建议跑

把 vLLM 与 SGLang 的服务端参数与客户端口径逐项对齐后，用同一个压测客户端先后压两边，产出含结论反转条件的选型报告

四格数字（两个引擎 × 关缓存/开缓存）均为实测；SGLang 开缓存后吞吐 3248 → 6069 接近翻倍，vLLM 侧前缀命中 87.4%，两侧命中率口径不同、不做横向比较。

差异归因未采 kernel 时间线，改用命中率与吞吐对照；SGLang 侧命中率读数受采样口径限制不可用，两侧命中率不做横向比较。

项目 1.3 **推理系统架构设计与容量规划** **实操** 必跑（2x4090）

用 kv_planner.py 手算 KV Cache 显存上界后压测饱和 QPS，产出含分层架构与采购测算的容量规划文档

压测饱和点实测 18.86 QPS（客服画像 in 900 / out 200、并发 32），调优前 16.71，容量与采购测算按 18.86 QPS 反推。

调优组同时打开前缀缓存并把单步 token 预算由 2048 提到 8192，收益不单独归因其中一项。

2 大模型芯片与集群硬件底座基础

GPU 执行模型、存储金字塔、Roofline 模型与互联网络；一个 kernel 受算力限制还是受带宽限制，由它的算术强度是否达到硬件的算力带宽比决定。

2.1 GPU 硬件体系架构与执行模型

- 2.1.1 SIMT 执行模型：GPU 与 CPU 的根本区别
- 2.1.2 SM 内部结构
- 2.1.3 Tensor Cores 协同
- 2.1.4 Warp 调度与 Occupancy

2.2 存储金字塔与内存墙瓶颈

- 2.2.1 存储层次的基本原理：局部性
- 2.2.2 GPU 存储层次
- 2.2.3 HBM 与片上 SRAM 的带宽与时延对比
- 2.2.4 内存墙瓶颈
- 2.2.5 虚拟内存映射开销

2.3 Roofline 模型

- 2.3.1 算术强度定义
- 2.3.2 Roofline 模型
- 2.3.3 计算受限与带宽受限的边界评估方法
- 2.3.4 Roofline 模型与算子算术强度评估 代码

2.4 芯片内与跨节点互联网络

- 2.4.1 互联层次
- 2.4.2 PCIe 吞吐
- 2.4.3 NVLink 与 NVSwitch 拓扑
- 2.4.4 InfiniBand 与 RoCEv2 网卡架构
- 2.4.5 GPUDirect RDMA

2.5 硬件原生数值格式与算力演进：E4M3FN 只有一个 NaN 编码，按 IEEE 754 惯例读会把最大值 448 判成 NaN

- 2.5.1 浮点数编码原理

2.5.2 数值格式与精度-范围对比
2.5.3 FP16/BF16 硬件特性
2.5.4 INT8/INT4 累加器
2.5.5 原生 FP8 规范
2.5.6 OCP 微缩量化 MX 格式底层支持
2.5.7 算力演进总览
2.5.8 FP8/FP4 浮点编码与精度范围分析 代码

2.6 **Triton 入门与 GPU 编程模型：算子融合减少的是 HBM 访存，不是计算量**

2.6.1 为什么需要 Triton
2.6.2 GPU 并行执行模型：Grid / Block / Warp / Thread
2.6.3 Triton 的抽象层次与编程模型
2.6.4 Triton 核心语法速查
2.6.5 向量加法（Triton Hello World） 代码
2.6.6 Softmax 算子（2D pid 寻址实战） 代码
2.6.7 用 @triton.autotune 自动调优 Softmax 代码

章末产出 脚本 9 报告 4 结构化数据 10 日志 12 profile 产物 7

项目 2.1 **vLLM/SGLang Kernel 性能基准套件** **实操** 必跑（2x4090）

从理论上限手算与 CUDA Event 计时，到 PagedAttention V2、FlashAttention、FlashInfer 三个后端的 kernel 实测，搭一套性能基准套件

H100 上以 CUDA Event 实测四个 attention 后端（BS=32、KV=1024）：decode 侧 FlashInfer 0.0543 ms、PagedAttention V2 0.0969 ms，同口径下差 1.8 倍，选后端的收益大于调参。

三个后端同在 vLLM 侧环境，SGLang 侧未复测；结果表中 3.3 / 105.6 TFLOPS 两行为推导上界。

项目 2.2 **Roofline 性能分析平台** **实操** 必跑（需 ncu 权限）

用 ncu 采集 kernel 实测数据并落到 Roofline 模型上，判定每个 kernel 是带宽受限还是算力受限

cuBLAS GEMM（nvjet）一类采不到 FLOPs，只按 DRAM 字节做带宽诊断。

项目 2.3 **自定义 Triton 融合算子开发** **从零实现** 必跑（需 ncu 权限）

用 Triton 从零手写 Bias+SiLU 融合算子，用 ncu 的 DRAM 字节列证明访存真降了并算出同口径加速比；另实现 Online Softmax，解决分块规约下的数值正确性问题

ncu 实测 Bias+SiLU 融合：DRAM 访存降 48.65%（推导预期 50%）、耗时 41.18 → 22.37 us，两者降幅同量级，收益来自减少 HBM 访存而非提升算力；Online Softmax 与 torch.softmax 对拍误差 1.86e-09。

GPU 型号与驱动版本未记录；缩小 BLOCK_N 暴露 naive softmax 局部最大值错误这一步无实测值；RMSNorm + 量化为自选扩展，不含实测。

3 自回归解码与推理瓶颈

算力、带宽、容量、通信四类瓶颈分别什么时候出现，本篇各章对应哪一类；Prefill 计算密集、Decode 访存密集，这个判断手算算术强度就能定，不必先跑。

3.1 **自回归生成过程**

3.1.1 从联合概率到逐 Token 生成：为什么必须一个一个来

3.1.2 一步生成在模型里走过的路

3.1.3 上下文窗口与生成长度：一笔要算清的预算

3.1.4 用真实词表观察 Token 代码

3.1.5 手写一个最小的贪心解码循环 代码

3.2 **Prefill 与 Decode 两个阶段：Decode 每步的计算量并不恒定，位置越靠后开销越大**

3.2.1 Prefill 阶段（预填充）

3.2.2 Decode 阶段（逐 Token 解码）

3.2.3 两阶段的延迟画像：TTFT 与 TPOT

3.2.4 每步的计算量并不恒定：位置越靠后开销越大

3.2.5 错误示例先行：把两条估算式用反会怎样

3.3 **算术强度分析：吞吐上界能手算，不必先跑**

3.3.1 Prefill 与 Decode 的算术强度与利用率对比 代码

3.3.2 从算术强度到吞吐上界：一条估算式

3.3.3 并行形态与批量效应的 CPU 亲手验证 代码

3.4 **四类瓶颈与本篇路线图**

3.4.1 算力与带宽：Prefill 是计算密集，Decode 是访存密集

3.4.2 容量：显存占用的四项构成与不够时的抢占

3.4.3 通信：多卡分摊了权重，每步多出串行通信开销

3.4.4 判定受限于哪一类：先分清三种利用率口径，再下结论

3.4.5 本篇路线图：每一章对应哪一类瓶颈

章末产出 脚本 9 报告 3 结构化数据 23 日志 22 profile 产物 3

项目 3.1 **Prefill/Decode 算术强度实测与可视化** **实操** 必跑（2x4090）

用一阶模型算出两阶段算术强度并画上 Roofline，再与 vLLM/SGLang 实测对照，把decode 受带宽限制变成可复算的数字

batch 256 一档因 Prefill 激活装不下未测，堆 batch 有效的结论只覆盖 8 到 64，且只在 SGLang 一侧。

项目 3.2 **Decode 阶段 GPU 利用率诊断** **实操** 建议跑

压一个短输入长输出的纯 decode 负载，先用 nvidia-smi 量出利用率再说清它与忙闲比、SM active 的差别，然后只抬并发上限做一次同口径 A/B，按预先登记的判定标准判定本次调优不成立

nvidia-smi 利用率 5.4% → 7.3%；抬并发上限后 ITL P99 由 10.86 升至 14.08 ms，按预先登记的判定标准，本次调优判定不成立。

nsys 与 ncu 未采集，低利用率的根因是依据 smi 与引擎默认参数提出的假说，尚未证伪；强制 eager 的反向对照未覆盖。

项目 3.3 **低延迟场景 Prefill-Decode 联合调优** **实操** 建议跑

在 TTFT/TPOT 双门槛 SLA 下网格搜索 chunked prefill 等参数，产出配置清单

实跑是四格缩网格，每格一轮，推荐配置未复跑。

第二部分　单卡显存与批处理

4　KV Cache 机制与结构变体

KV 占多少、怎么省：显存怎么算，MQA / GQA / MLA 各省在哪一项，稀疏化淘汰的边界在哪。

4.1　为什么需要 KV Cache

- 4.1.1　自回归生成的重复计算问题
- 4.1.2　Attention 中的重复计算
- 4.1.3　KV Cache 的原理：缓存历史，避免重复计算
- 4.1.4　KV Cache 的数据流详解
- 4.1.5　为什么只缓存 K 和 V，不缓存 Q
- 4.1.6　有无 KV Cache 的计算量对比：$O(n^2)$ 降到 $O(n)$
- 4.1.7　KV Cache 的代价：显存开销

4.2　经典 KV Cache 的显存计算

- 4.2.1　KV Cache 显存计算器与架构对比　代码

4.3　MQA 与 GQA 机制

4.4　MLA（低秩压缩注意力机制）

- 4.4.1　MLA 的核心思想
- 4.4.2　MLA 的计算流程
- 4.4.3　MLA 的 RoPE 兼容处理
- 4.4.4　MLA 推理时的投影矩阵吸收
- 4.4.5　MLA vs GQA/MQA 对比
- 4.4.6　MLA 前向计算与矩阵吸收实现　代码

4.5　长文本 KV Cache 稀疏化与逐退（Eviction / Attention Sink）

- 4.5.1　Attention Sink 现象
- 4.5.2　StreamingLLM
- 4.5.3　H2O（Heavy-Hitter Oracle）
- 4.5.4　KV Cache 逐退策略对比
- 4.5.5　StreamingLLM 与 H2O 逐退策略实现　代码

4.6 **KV Cache 的生命周期管理**

章末产出 脚本 5 报告 3 结构化数据 22 日志 14 profile 产物 2

项目 4.3 **长文本场景 KV Cache 淘汰策略实现与 A/B 测试** **从零实现** 读透即可

在离线 harness 中实现 StreamingLLM / H2O 等淘汰策略并评精度，再量化KV Cache 减半在引擎侧能兑现多少收益

离线精度四组跑在自建夹具上而非官方 LongBench，困惑度绝对值不作结论，生产淘汰预算未选出。

5 Flash Attention 与 Decode 算子

Flash Attention 的 FLOPs 与标准 Attention 相同，省的是 HBM 访问；分块计算与在线 Softmax 是它的数学核心，Chunked Prefill 与 Flash Decoding 是推理侧的两个变体。

5.1 **标准 Attention 的问题**

5.2 **标准 Attention 的显存与计算分析**

5.3 **Flash Attention 的核心思想：分块计算（Tiling）**

5.4 **Flash Attention 分块计算详解**

5.4.1 Flash Attention 分块计算（Triton 实现） 代码

5.5 **在线 Softmax（Online Softmax）：Flash Attention 的数学核心**

5.5.1 在线 Softmax 分块更新算法实现 代码

5.6 **Flash Attention 与标准 Attention：显存对比**

5.7 **Flash Attention 与标准 Attention：HBM 访问对比**

5.8 **计算量对比与 Wall-clock 加速**

5.9 **Flash Attention 2 与 Flash Attention 3 的改进**

5.10 **Flash Attention 在推理中的特殊应用**

5.11 **标准 Attention 与 Flash Attention 逐项对比：FLOPs 相同，省的是 HBM 访问**

5.12 **Chunked Prefill（分块首字预填）**

5.12.1 Chunked Prefill 的核心问题

5.12.2 Chunked Prefill 的调度策略

5.12.3 Chunked Prefill 的 KV Cache 拼接机制

5.12.4 Chunked Prefill 的性能权衡

5.12.5 Chunked Prefill 调度器实现 代码

5.13 **Flash Decoding 与 Flash Decoding++**

5.13.1 Decode 阶段的特殊性

5.13.2 Flash Decoding

5.13.3 Flash Decoding++

5.14 **算子融合：FlashPagedAttention**

5.14.1 为什么需要融合

5.14.2 FlashPagedAttention 的实现要点

5.15 **硬件层优化：CUDA Graphs 机制**

5.15.1 Decode 阶段的 Kernel Launch 开销

5.15.2 CUDA Graphs 原理

5.15.3 CUDA Graphs 在推理系统中的应用

章末产出 脚本 15 报告 8 结构化数据 62 日志 46 profile 产物 2

项目 5.1 **vLLM Attention 后端性能对比与选型** **实操** 建议跑

对 FLASH_ATTN / FLASHINFER / TRITON 等后端做同口径压测，产出分场景的 attention 后端选型结论

预筛的五个后端里 FLEX_ATTENTION 因过慢跳过，实测四个；实跑每档 64 条、并发 32，小于设计规模的 400 条。

项目 5.2 **生产流量画像下的 Chunked Prefill 调优** **实操** 建议跑

在长短混合的流量画像下网格搜索分块大小，定出短请求的达标配置，并用四档实测说明长文档那一段没有一档能同时守住 TTFT 与 ITL 两条 SLA、只能分池

长文档段四档分块的 TTFT 实测全部超出 SLA，结论是长短流量分池；长请求单独对待实测后短、长同时劣化，按预先登记的判定标准丢弃。

压测用合成的长短两段长度分布、非同批次内混合，实跑短 80 条 / 长 32 条，小于设计规模的 420 / 180 条；瓶颈定位未采 nsys / ncu，改用四档网格数字交叉验证。

项目 5.3 **SGLang FlashInfer Decode 后端长上下文实测** **实操** 建议跑

在 32K 长上下文、低并发下实测只换 FlashInfer decode 后端相对默认 fa3 有没有收益，结论是没有：三条路径吞吐贴齐，瓶颈在带宽与调度

实测吞吐 202.5 → 202.0 tok/s，收益为零。

并行归约 kernel 是否真正出现未采 nsys 确认，只有启动日志证明后端字段生效。

6 PagedAttention 与显存管理

分页 KV 管理的机制、Attention 计算方式与优势边界，并列出 vAttention 等替代路线。

6.1 **传统 KV Cache 管理的问题**

6.2 **PagedAttention 核心思想**

6.3 **PagedAttention 的 Attention 计算**

6.3.1 PagedAttention 块表与显存池化管理 代码

6.4 **PagedAttention 的优势**

6.5 **vAttention 等替代方案**

6.6 **在锁定版本引擎里怎么用**

6.6.1 vLLM 0.26.0

6.6.2 SGLang 0.5.16

6.6.3 启动阶段两条与 KV 池有关的报错

章末产出 脚本 8 报告 9 结构化数据 59 日志 54

项目 6.1 **精读 vLLM BlockManager 并复现带 COW 的分页管理器** **从零实现** 必跑（纯 CPU）

精读 vLLM BlockManager 源码后用纯 CPU 复现带写时复制的分页 KV Cache 管理器并验算块表

纯 CPU 实跑：共享前缀的物理块数据一致性误差 0.00e+00；补写时复制之前先复现出块内污染，补上之后块表与引用计数逐项复算得上。

项目 6.2 **显存碎片率监控与告警系统** **实操** 建议跑

三档 block_size 压测加 vLLM 在线 KV Cache 采样，给出差 5% 内且无抢占就选默认 16 的依据

表中碎片率是尾块期望公式算出来的，不是在线采样值。

项目 6.3 **vLLM PagedAttention 与 SGLang RadixAttention 前缀复用对比** **实操** 读透即可 案例解读

同卡分时起 vLLM 与 SGLang，先测 APC 开关下的前缀重叠斜率，再做粒度对齐与各自默认两组跨引擎对比，产出写明反转条件的选型口径

跨引擎对比已补齐，交付粒度对齐与各自默认两张表。

两组实测均未跑满设计口径（设计为每档 500 条、3 轮取中位数；前缀斜率组每档 1 轮，跨引擎组每档 48 条请求、并发 32），绝对值只看趋势，不作通用基准。

7 Continuous Batching

Continuous Batching 是调度层技术，不是硬件执行层技术；静态批处理的问题、迭代级调度的核心思想，以及 Prefill 与 Decode 的混合调度。

7.1 **静态批处理的问题**

7.2 **Continuous Batching 核心思想**

7.3 **Prefill 与 Decode 的混合调度**

7.3.1 Continuous Batching 调度循环 代码

7.4 **Continuous Batching 是调度层技术，不是硬件执行层技术**

7.4.1 GPU 真正的执行单位

7.4.2 batch 如何映射到硬件

7.4.3 为什么 Continuous Batching 属于调度层

7.4.4 batch_size 是软件概念，warp 才是硬件概念

章末产出 脚本 12 报告 6 结构化数据 53 日志 39 profile 产物 3

项目 7.2 **Batching 参数压测调优** **实操** 建议跑

对 `max_num_seqs` × `max_num_batched_tokens` 做网格压测，找出 SLA 约束内吞吐最高的组合并说清为什么是它

每格两轮、每轮 48 至 64 条，小于设计规模的三轮与 1000 条；推荐格 128 未复跑。

项目 7.3 **差异化 SLA 的 Batching 策略** **实操** 建议跑

在同一服务上同时压实时与批量两类流量，用 goodput 与 TTFT 量出优先级调度相对先来先服务的真实收益幅度，并写明这套策略在什么条件下会失效

先来先服务基线两类已各自 100% 达标，优先级调度为边际改善：实时首包 P99 160 → 150 ms。

压测规模为实时 80 条 + 批量 32 条，设计规模为 300 / 120 条；失效条件为推理所得、未另造负载实测；服务端两路全量采集未覆盖，瓶颈定位用一段复现 prefill 与 decode 交织形态的最小程序时间线。

第三部分 分布式并行与通信

8 数据并行（DP）演进：DDP、ZeRO 与 FSDP

DDP → ZeRO → FSDP 三者的显存节省与通信代价，以及选型依据。

8.1 **DDP（DistributedDataParallel）：经典数据并行**

8.2 **ZeRO（Zero Redundancy Optimizer）：分片消除冗余**

8.2.1 ZeRO-1：优化器状态分片

8.2.2 ZeRO-2：梯度也分片

8.2.3 ZeRO-3：参数也分片

8.3 **FSDP（Fully Sharded Data Parallel）：PyTorch 原生 ZeRO-3**

8.4 **三者关系与选型**

8.5 **实战代码**

8.5.1 DDP 完整训练示例

8.5.2 DeepSpeed ZeRO 训练示例

8.5.3 FSDP 完整训练示例

8.5.4 显存与通信监控

8.5.5 三种方案启动方式对比

章末产出 脚本 4 报告 4 结构化数据 13 日志 24 profile 产物 2

项目 8.1 **DDP 梯度同步开销剖析与 bucket 调优** **实操** 读透即可

在 2×H100 上用 DDP 四计时器与 nsys 双口径剖析梯度 AllReduce 开销，只改 bucket_cap_mb 一处，按预先登记的判定标准决定默认值该不该动

默认配置下桶数固定在 17，调整 bucket_cap_mb 判定为改动不成立；暴露通信 5.2% → 2.3% 的收益只出现在隐藏层 2048 的教学对照上。另有纯 CPU 两进程路线已实跑。

本项目采集 nsys、不采集 ncu。

项目 8.2 / 8.3 **DDP → ZeRO-2 → ZeRO-3 显存对照** **实操** 读透即可 案例解读

用 FSDP 的 SHARD_GRAD_OP / FULL_SHARD 做 DDP → ZeRO-2 → ZeRO-3 的等价显存对照，量出每档单卡峰值显存与墙钟代价，并解释 ZeRO-3 为什么未再节省

单卡峰值显存 DDP 19.6 GiB → FSDP 8.2 GiB（约 2.4 倍），ZeRO-3 相对 ZeRO-2 未再节省，墙钟 0.924 → 3.108 秒。

DeepSpeed 在参考环境不可用，ZeRO-2 / ZeRO-3 以 FSDP 的 SHARD_GRAD_OP / FULL_SHARD 等价代跑；只量显存与墙钟，AllReduce 通信量未测；交付结论表，不含分步命令与排错。

9 张量并行（Tensor Parallelism）

线性层 / MLP / Attention 的切分方式与通信开销：重叠的上限就是通信时间本身，拆分不减总量，TP 不出节点；再延伸到序列并行与上下文并行。

9.1 **基本原理：并行地算一层，代价是每层前向都多一次 AllReduce**

9.2 **线性层的切分**

9.2.1 列切分（Column Parallel）

9.2.2 行切分（Row Parallel）

9.3 **MLP 层的 TP 并行**

9.4 **Attention 层的 TP 并行**

9.5 **TP 的通信开销**

9.5.1 张量并行线性层切分（Column/Row Parallel） 代码

9.6 **通信开销与优化：重叠的上限就是通信时间本身，拆分不减总量**

9.6.1 All-Reduce 的通信量分析

9.6.2 三条优化路径：重叠上限是通信时间，拆分不减总量，TP 不出节点

9.7 **序列并行（Sequence Parallelism）**

9.7.1 Megatron-LM 序列并行

9.7.2 Ulysses 序列并行

9.7.3 Ring Attention

9.7.4 序列并行方案对比

9.8 **Context Parallelism（上下文并行）**

9.8.1 CP 与 SP 的区别

9.8.2 CP 的 Attention 计算流程（Ring Attention 实现）

9.8.3 CP 在推理阶段的应用

9.8.4 CP 与其他并行的组合

章末产出 脚本 5 报告 3 结构化数据 22 日志 43 profile 产物 2

项目 9.1 **vLLM 张量并行部署的一致性与开销验证** **实操** 读透即可

实测 vLLM 张量并行下单卡与多卡的贪心输出何时分歧、为何分歧，量出 TP=2 相对单卡 1.50× 的吞吐与两种改法的代价，产出取舍表

两卡合成 AllReduce 对照实测通信占比 89.2%，该数来自合成对照、不是 vLLM 服务实采，用来解释 TP=2 为什么只拿到 1.50× 而不是 2×。

该数不是从 vLLM 服务上实采；TP=4 只跑到冷启动、不计加速比；序列并行与上下文并行为合成注意力对照。

10 流水线并行（Pipeline Parallelism）

气泡问题、微批次与 1F1B 及交错式调度、DualPipe，以及 PP 在推理侧的特殊考量。

10.1 **基本原理**

10.2 **气泡问题：推理侧的气泡与训练侧的 1F1B 气泡成因不同**

10.3 **微批次流水线（Micro-batch Pipeline）**

10.4 **1F1B 调度**

10.4.1 训练中的 1F1B 调度

10.4.2 推理中的 1F1B

10.4.3 1F1B 流水线调度模拟 代码

10.5 **交错式 1F1B（Interleaved 1F1B）**

10.5.1 动机：1F1B 的气泡瓶颈

10.5.2 虚拟 Stage 与 Chunk 切分

10.5.3 交错调度时序图

10.5.4 气泡与通信开销分析

10.6 **DualPipe 双向流水线并行**

10.6.1 动机：前向与反向的负载不对称

10.6.2 双向流水线核心思想

10.6.3 DualPipe 调度时序图

10.6.4 DualPipe 与 1F1B / ZB1P 对比

10.7 **PP 在推理中的特殊考量：推理只有前向，1F1B 与交错式调度的前提不成立**

章末产出 脚本 3 报告 1 结构化数据 17 日志 22 profile 产物 2

项目 10.1 **vLLM Pipeline Parallel 部署与 stage 利用率可视化** **实操** 读透即可

vLLM 起 PP=2 实测各 stage 忙碌占比与 TTFT，用时间轴回答推理侧 PP 气泡与训练侧 1F1B 是否同一回事

通信占比来自 torch profiler 事件，含对端未就绪的等待，不是纯 P2P kernel 时间。

11 专家并行（Expert Parallelism）

MoE 结构、All-to-All 通信与负载均衡问题，并给出六种并行策略的组合关系。

11.1 **MoE 模型结构**

11.2 **专家并行原理**

11.3 **All-to-All 通信**

11.4 **Expert Parallelism 的负载均衡问题**

11.4.1 MoE 前向与负载均衡损失 代码

11.5 **六种并行策略：DP / TP / PP / SP / EP / CP 的关系与组合**

11.5.1 六种并行策略的切分维度总览

11.5.2 六种并行的关系图

11.5.3 并行策略的约束与兼容性矩阵

11.5.4 典型大规模系统的并行组合方案

11.5.5 并行策略选择决策树

11.6 **现代并行与 MoE 极致重叠（以 DeepSeek-V3 为例的系统工程）**

11.6.1 DualPipe 调度算法：计算-通信完全重叠

11.6.2 无辅助损失负载均衡（Auxiliary-Loss-Free Load Balancing）

11.6.3 延伸：DeepSeek-V4 之后，这条路线走到了哪里

章末产出 脚本 7 报告 8 结构化数据 36 日志 38

项目 11.1 **SGLang 部署 MoE 并解析专家路由分布** **实操** 读透即可

SGLang 起 MoE 后 dump 专家路由，解析成逐层 MaxVio 不均衡度表，把均衡与否变成数字

不均衡的算力代价未量化，结论只来自路由记录与两档吞吐。

项目 11.2 **EP 部署下的 All-to-All 通信压测** **实操** 读透即可 案例解读

拆开 EP 规模与负载倾斜两个变量做 All-to-All 纯通信压测，算出扩 EP 会同时放大专家热点，给出纯通信口径下的扩容建议

真机通信只扫到 EP=2（单机两卡）；EP 越大专家热点越重，不均衡度 EP=4 为 3.19、EP=8 为 5.44，这两个是纯 CPU 簿记结果、不是真机耗时。

EP=4 / EP=8 的不均衡度 3.19 / 5.44 为纯 CPU 簿记结果、不是真机耗时；端到端吞吐与开启 EPLB 的对照因引擎适配未完成，无数据。

项目 11.3 **复现无辅助损失负载均衡策略** **从零实现** 必跑（纯 CPU）

纯 CPU 复现 MoE 的 Loss-Free Balancing 路由，同种子对比多种均衡策略的 MaxVio 与路由坍缩

五组策略同种子对照实测：不做均衡时最热专家吃 2051 个 token、最冷的一个都没拿到；偏置法把 MaxVio_global 压到 0.162，比辅助损失的 0.576 低 72%，且 eval_loss 更低。

项目 11.5.2 **vLLM 3D 并行（TP + PP + DP 副本）部署扩展性实测** **实操** 读透即可

在 8 卡上把 TP×PP×副本数的五种拆法按同一压测口径各测一轮，用吞吐、TTFT、显存三组数据分出最优组合与切分过细、收益转负的区间

最优组合在总并发 64 一档上验证通过；最优是条件解不是常数，换输入输出长度、目标并发或延迟约束都要重扫。

实测模型为 Qwen3-8B，课程给定的模型为 DeepSeek-V2-Lite；TP=2 两组需加 enforce-eager 才能启动，与其余三组服务参数不一致；通信与气泡未做采样器拆分。

12 通信原语（Communication Primitives）

TP=2 吞吐低于单卡常常是预期结果不是故障：通信开销确定要付，算力收益不一定拿得到。本章从 Send / Recv 讲到集合通信原语与 Ring AllReduce，再落到 NCCL 与 TP / PP / EP 的映射。

12.1 **通信原语概述**

12.2 **点对点通信：Send / Recv**

12.3 **集合通信原语详解**

- 12.3.1 Broadcast（广播）
- 12.3.2 Scatter / Gather
- 12.3.3 Reduce / AllReduce
- 12.3.4 ReduceScatter
- 12.3.5 AllGather
- 12.3.6 All-to-All

12.4 **Ring AllReduce：AllReduce 的高效实现**

- 12.4.1 Ring AllReduce 原理
- 12.4.2 使用 PyTorch 实现底层通信原语 代码
- 12.4.3 Ring AllReduce 的通信量分析
- 12.4.4 Ring AllReduce 的延迟分析
- 12.4.5 用 PyTorch P2P 原语手写 Ring AllReduce 代码

12.5 **通信原语的实现与优化**

- 12.5.1 NCCL（NVIDIA Collective Communications Library）
- 12.5.2 通信与计算重叠
- 12.5.3 其他通信优化技术

12.6 **通信原语与并行策略的完整映射**

12.7 **通信原语选择决策**

12.8 **NCCL 深度解析**

12.8.1 NCCL 整体架构

12.8.2 Ring 算法

12.8.3 Tree 算法

12.8.4 CollNet

12.8.5 SHARP 网络内聚合

12.8.6 NCCL 拓扑探测

12.8.7 GPU 拓扑探测与 P2P 带宽矩阵检测 代码

12.8.8 NCCL 环境变量

12.8.9 NCCL 环境变量矩阵化压测（自动化选型） 代码

12.8.10 NCCL Profiling

12.8.11 解析 NCCL_DEBUG=INFO 日志提取关键指标 代码

12.8.12 通信优化最佳实践

12.8.13 NCCL Benchmark 代码

章末产出 脚本 5 报告 2 结构化数据 11 日志 24 profile 产物 4

项目 12.2 **NCCL 环境变量矩阵化压测工具** **实操** 建议跑

外层驱动逐组合起全新进程矩阵化压测 NCCL 环境变量，用 AllReduce 带宽逐组验证设置真被采纳

冠军 NCCL_BUFFSIZE 未复测，8M 档的增益落在噪声带内。

项目 12.3 **vLLM 多卡部署下的 NCCL 通信瓶颈定位** **实操** 读透即可

用 NCCL 日志核验两卡是否走 GPU 直连，再以 TP=1 / TP=2 同口径压测算出多卡的实际收益与通信代价，并搭一张带通信域快照的常驻健康看板

已实测通信路径核验（两卡直连、24 条通道）与 TP=1 / TP=2 / 关自定义 AllReduce 三档同口径压测；7B 放得下单卡时 TP=2 吞吐低于单卡是预期结果，不是故障。

通信核占 GPU 时间的比例受实跑引擎版本限制未采到，本项目不给该比例。

第四部分　解码加速与调度

13　投机解码（Speculative Decoding）

投机解码为什么不改变输出分布、Medusa / EAGLE / MTP 三条路线怎么选，以及为什么接受率高不等于快：实测接受长度从 2.14 升到 3.16，吞吐仍低于不开投机的基线。

13.1 **核心动机**

13.2 **算法流程**

13.3 **分布不变性的数学保证：草稿被拒绝时，接受-拒绝采样仍保证输出分布不变**

13.4 **加速比分析：接受率高不等于快**

13.4.1 用 NumPy 模拟投机解码的接受-拒绝采样与接受率口径 代码

13.5 **投机解码的变体**

13.5.1 自投机解码（Self-Speculative Decoding）

13.5.2 树形投机解码（Tree Speculative Decoding）

13.5.3 树形注意力（Tree Attention）矩阵详解

13.6 **轻草稿模型投机解码（Medusa 与 EAGLE）**

13.6.1 Medusa

13.6.2 EAGLE

13.6.3 Medusa 多头投机解码实现 代码

13.7 **启发式/检索投机解码（Prompt Lookup Decoding）**

13.7.1 Prompt Lookup Decoding

13.7.2 投机解码方案对比

13.8 **MTP（Multi-Token Prediction）投机解码**

13.8.1 MTP 与 Medusa/EAGLE 的核心区别

13.8.2 MTP 融合层的工程实现

13.8.3 MTP 的推理流程（投机解码模式）

13.8.4 MTP 的训练模式与推理模式

13.8.5 投机解码方案综合对比

13.9 **使用 SpecForge 训练 EAGLE3 并用 SGLang 部署与吞吐测试** 代码

13.9.1 EAGLE3 工作原理回顾

13.9.2 安装 SpecForge 与 SGLang：训练 1 张卡起（8B online 训练建议 8 卡），部署 1 张即可

13.9.3 训练数据统一成 ShareGPT 风格 jsonl，越贴近目标分布接受率越高

13.9.4 训练 EAGLE3 Draft 模型

13.9.5 用 SGLang 部署 EAGLE3 投机解码

13.9.6 吞吐量与延迟测试

13.9.7 调优与排错指南

章末产出 脚本 9 报告 4 结构化数据 19 日志 33

项目 13.1 **在 SGLang 中启用投机解码并验证接受率与加速比** **实操** 建议跑

在 39 通真实客服对话上扫四档草稿长度，量出接受长度从 2.14 升到 3.16、吞吐却始终低于不开投机的基线，给出该不该开投机的口径

并发 8 与 32 两档实测完成；投机档要把静态显存占比由 0.80 降到 0.50 才起得来，两侧不完全同口径。

实测所用引擎无自适应投机开关，改动一处换成缩小草稿树；投机档显存不足，两侧不完全同口径。

项目 13.3 **用 SpecForge 训练 EAGLE3 并用 SGLang 部署压测** **实操** 读透即可

为目标模型专训一个 EAGLE3 草稿头，部署后与现成小模型草稿同口径对比加速比

训练走在线加 HF 目标模型的路线，离线抓隐状态与离线训练两节未执行。

14 Sampling 与生成策略

温度在确定性与均匀分布之间对熵做连续插值，Gumbel-max 把采样变成 argmax；采样本身有开销：一个批级 LogitsProcessor 写得好坏，TPOT 开销差在 +10.8% 与 +0.06% 之间。

14.1 Sampling 基础

14.1.1 Greedy Decoding（贪心解码）

14.1.2 Random Sampling（随机采样）

14.2 Temperature Sampling

14.2.1 Temperature 原理

14.2.2 温度采样的推导：温度在确定性与均匀分布之间对熵做连续插值

14.2.3 温度对生成质量的影响

14.2.4 Temperature Sampling 代码

14.3 Top-k Sampling

14.3.1 Top-k 原理

14.3.2 Top-k 与 Greedy 对比

14.3.3 Top-k Sampling 代码

14.4 Top-p（Nucleus Sampling）

14.4.1 动态候选集

14.4.2 Top-p 与 Top-k 对比

14.4.3 Top-p Sampling 代码

14.5 **Typical Sampling**

14.6 **Min-p Sampling**

14.7 **重复惩罚与频率控制**

14.7.1 Repetition Penalty

14.7.2 Frequency Penalty

14.7.3 Presence Penalty

14.8 **Logits Processor：同一条规则，写成批级还是逐条，开销差在两个数量级**

14.8.1 五类约束型 Processor

14.8.2 Logits Processor Pipeline 代码

14.9 **Sampling Kernel：Gumbel-max 把采样变成 argmax**

14.9.1 高效 GPU 采样流水线

14.9.2 Gumbel-max 原理：把采样变成 argmax

14.9.3 GPU Sampling Triton 实现 代码

章末产出 脚本 8 报告 3 结构化数据 16 日志 8

项目 14.2 **自定义 LogitsProcessor 做业务合规过滤** **从零实现** 建议跑

金融文案合规场景下离线构封禁表、写一个批级 LogitsProcessor 在采样前封死违禁词，并把它的 TPOT 开销从 +10.8% 压到 +0.06%

拦截正确性四条用例与三档压测均为实测：同一份封禁表，逐条写法 TPOT 开销 +10.8%，改成批级一次处理只剩 +0.06%，差别全在写法。

torch profiler 轨迹未落盘，“开销出在 kernel 启动次数”这一归因由代码形态加同口径复测反证，无 GPU 时间线证据；与引擎内置 bad_words 的对照在这条诱导 prompt 下两者都拦住了，“内置方案会漏词”只在离线状态机自检中复现过、服务端未复现。

项目 14.3 **内置 GPU Sampling 压测与瓶颈分析** **实操** 建议跑

阶梯加压测出采样占每 token 的成本占比，用 TPOT 与 FlashInfer 后端对照回答它是不是吞吐瓶颈

采样占比随并发上升，16.7% 只对并发 64 成立；SGLang 侧未交叉验证。

15 量化技术

GPTQ / AWQ / SmoothQuant / FP8 与 KV Cache 量化，以及 MX 格式的硬件-软件协同设计；AWQ 与 GPTQ 在引擎里走同一族 kernel，选哪个的差别小于换一档 batch 规模的差别。

15.1 **量化的基本原理**

15.2 **量化分类**

15.3 **常见量化方案**

15.3.1 量化与反量化基础（INT8/INT4 对称量化） 代码

15.3.2 Triton 实现线性量化与反量化 Kernel 代码

15.4 **GPTQ**

15.4.1 GPTQ 问题建模：目标是量化误差对输出的影响最小，等价于逐行优化

15.4.2 OBS 最优补偿原理

15.4.3 补偿公式推导：Hessian 精确，近似只在用逐层误差代替全网络损失

15.4.4 Cholesky 分解加速

15.4.5 GPTQ 的分组量化

15.4.6 GPTQ 实践要点

15.4.7 GPTQ 逐列量化与误差补偿 代码

15.5 **AWQ**

15.5.1 量化误差的影响是权重误差乘激活幅度，故保护大激活对应的权重通道

15.5.2 缩放因子优化

15.5.3 最优缩放因子搜索

15.5.4 AWQ 与 GPTQ 对比

15.5.5 AWQ 激活感知权重量化 代码

15.6 **SmoothQuant**

15.6.1 SmoothQuant 激活平滑迁移 代码

15.7 **量化感知训练（QAT）**

15.7.1 QAT 的思路：把量化放进训练的前向，用反向传播补偿它带来的误差

15.7.2 伪量化（Fake Quantization）

15.7.3 直通估计器（STE）

15.7.4 QAT 训练流程

15.7.5 QAT 的关键技巧

15.7.6 QAT 与 PTQ 对比

15.7.7 可直接替换 nn.Linear 的 QAT 线性层 代码

15.8 **FP8 量化**

15.8.1 FP8 的两种编码格式

15.8.2 FP8 与 INT8 对比

15.8.3 FP8 推理流程

15.8.4 延迟缩放（Delayed Scaling）

15.8.5 FP8 量化的实践要点

15.8.6 FP8（E4M3/E5M2）量化与反量化 代码

15.8.7 FP8 真实 GEMM 推理（torch._scaled_mm）与 Fp8Linear 封装 代码

15.9 **KV Cache 量化**

15.9.1 KV Cache 量化的动机

15.9.2 KV Cache 量化的实现方式
15.9.3 KV Cache 量化的精度影响
15.9.4 KV Cache 量化的推理加速

15.10 **硬件-软件协同设计：FP4/FP6 与 OCP 微缩量化（MX 格式）**
15.10.1 从 FP8 到 FP4/FP6：硬件原生低精度的演进
15.10.2 FP4/FP6 的非均匀浮点编码
15.10.3 OCP MX（Microscaling Formats）规范
15.10.4 FP4/FP6 非均匀量化与反量化算子设计
15.10.5 FP4/FP6 在推理中的实际效果与精度权衡
15.10.6 NVFP4：NVIDIA Blackwell 的 FP4 变体

章末产出 脚本 16 报告 7 结构化数据 99 日志 32

项目 15.1 **三条量化路径的部署对比** **实操** 建议跑

同一模型的 bf16、在线 FP8、AWQ、GPTQ-Int4 四条路径同口径压测，按启动日志量权重与 KV 池，用困惑度和带区间的准确率测质量，产出写明适用边界的选型对照表

四路同口径压测完成，吞吐与权重显存是 W4A16 Marlin 的真值；AWQ 与 GPTQ 在 vLLM 里走同一族 Marlin kernel，两者差距小于同一方案换 batch 规模的差距。

官方 Qwen2.5-7B-Instruct-AWQ / GPTQ-Int4 两个仓库在实测机上无法下载，这两路改用 AutoAWQ 0.2.9 与 AutoGPTQ 0.7.1 对同一份本地 bf16 自行量化，困惑度与准确率含自建校准数据的差异、不等同官方量化版的质量；kernel 对照未采集，加速归因改用启动日志的显存占用加端到端吞吐；质量评测 50 题，四路的 95% 区间互相重叠，判不出谁更准。

项目 15.3 **FP8 权重 + FP8 KV Cache 端到端部署报告** **实操** 建议跑

FP8 权重与 FP8 KV Cache 做 2×2 正交实验，产出含显存占用算式与精度退档判据的权衡报告

精度评测用 50 题自建选择题集，不是课程给定的 C-Eval 500 题，四组准确率区间互相重叠。

16 前缀缓存与 RadixAttention

命中率单独一个数说明不了问题，省下的 Prefill 占比约等于命中率乘可复用前缀比例；本章讲静态前缀缓存与 RadixAttention、淘汰策略、哈希计算共享前缀的做法，以及 KV 复用在什么条件下失效。

16.1 **前缀缓存动机**
16.2 **静态前缀缓存**
16.3 **RadixAttention**
16.4 **KV Cache 淘汰策略**
16.4.1 RadixAttention 前缀树缓存 代码
16.4.2 命中率的口径与两个反直觉现象
16.5 **vLLM 的哈希计算共享前缀方法**
16.5.1 以 Block 为最小缓存单位：用一张哈希表替代基数树，查找 O(1)

16.5.2 哈希计算公式
16.5.3 vLLM 块级哈希链前缀共享的工作流程
16.5.4 vLLM 风格哈希链前缀缓存 代码
16.5.5 RadixAttention 与 vLLM Hash 对比

16.6 **KV 复用的正确性条件与失效场景**

16.6.1 什么时候前缀相同也不能复用
16.6.2 把命中率做成零的几种典型写法

16.7 **多实例部署与分层缓存**

16.7.1 前缀感知路由
16.7.2 分层卸载与何时划算

章末产出 脚本 10 报告 3 结构化数据 10 日志 43

项目 16.2 **vLLM APC 与 SGLang RadixAttention 命中率对比** **实操** 建议跑

同一份共享 System Prompt 流量分别压 vLLM APC 与 SGLang RadixAttention，只比同引擎开关缓存的 TTFT 相对改善，并在表里显式标出两边命中率口径不可比的格子

vLLM 侧前缀命中 78.7%；开缓存后首包 P99 反升（87.3 → 149.9 ms）而均值下降，SGLang 首包 P99 降 62%。交付的结论是“命中率不能当验收指标”。

SGLang 的命中率为区间瞬时表、压测窗口读数为 0，两侧命中率不排名；kernel 时间线未采集，定位改用首包与每 token 时延做正交。

项目 16.3 **多租户前缀缓存驱逐策略调优** **实操** 建议跑

按 64 租户 × 2048 前缀算出工作集与 KV 池容量比，缩池造出真实驱逐压力后扫容量与淘汰策略，用 TTFT P99 与池占用定取舍，产出容量换算式与失效条件

三组实测（小池 LRU、大池 LRU、大池 LFU）：容量翻倍使首包 P99 降 44.6%；同容量下 LFU 未胜 LRU，首包 P99 反高 5.3%。

SGLang 的命中率为区间瞬时表、三组读数均为 0，取舍判据用首包 P99 与 KV 池占用；第三档容量、vLLM 侧的容量扫描与 slru 一档未覆盖。

17 调度策略

vLLM 锁定版本里的抢占只剩 Recompute 一条路，Swap 换出路径已移除；本章讲请求级与迭代级调度、准入控制 Admission Control、抢占与多实例负载均衡路由。

17.1 **调度的核心目标与约束**

17.2 **请求级调度（Request-level Scheduling）**

17.2.1 FCFS（First Come First Served）
17.2.2 最短处理时间优先（SJF）
17.2.3 优先级调度（Priority Scheduling）
17.2.4 FCFS / SJF / 优先级调度对比 代码

17.3 **迭代级调度（Iteration-level Scheduling）**

17.3.1 Prefill 与 Decode 的混合调度策略

17.3.2 Chunked Prefill 与调度协同

17.4 **多实例负载均衡与路由策略**

17.4.1 轮询与加权轮询

17.4.2 最少连接与最少负载

17.4.3 一致性哈希：会话亲和与缓存局部性

17.4.4 随机与两次随机选择

17.4.5 负载均衡策略对比与选型

17.5 **并发控制与准入机制（Admission Control）**

17.5.1 并发数的约束因素

17.5.2 动态并发调整

17.6 **抢占（Preemption）**

17.6.1 抢占触发条件

17.6.2 换出到 CPU（Swap）

17.6.3 重计算（Recompute / RePrefill）

17.6.4 Swap 与 Recompute 的选择策略

17.6.5 Victim 选择策略

17.6.6 抢占式调度（Swap / Recompute） 代码

17.7 **高级调度策略**

17.7.1 前缀缓存感知调度（Cache-Aware Scheduling）

17.7.2 SLA 感知调度（SLA-Aware Scheduling）

17.7.3 Lookahead 调度（前瞻调度）

17.7.4 多模态感知调度（Multimodal-Aware Scheduling）

17.8 **一次请求经过的调度环节链路，与四个主流框架的取舍差异**

章末产出 脚本 6 报告 8 结构化数据 23 日志 47

项目 17.2 **抢占式调度：Recompute 与换出的现役形态** **源码精读** 建议跑

核实 vLLM 锁定版本里的 Preemption 只剩 Recompute 一条路，实测抢占的 TTFT 开销与触发条件

换出那一组实跑的是把权重卸到 CPU，不是 KV 换出；抢占开销只测到次数与尾延迟一层。

项目 17.3 **SLA 感知 + 前缀缓存感知的调度器** **从零实现** 建议跑

设计 SLA 分档与前缀亲和两条轴的调度策略，自写定档打分脚本，用加权违约横向比较三种策略：综合策略在一个引擎上最优、在另一个上输给轮询，学的是怎么判定该不该上线

两侧结论相反：SGLang 上前缀亲和 + 最长前缀匹配加权违约最低（582.2，对照轮询 771.2）；vLLM 上综合策略反而差于轮询。前缀副本数由 2.00 收敛到 1.00。

实测所用 SGLang 版本前缀感知排序与优先级调度不能同开，两条轴拆成三轮分开比，该侧单卡两实例显存不足、改为双卡各一实例；预先登记的“背景吞吐劣化不超过 10%”这道门禁无实测数填入，不给吞吐一侧的收益结论。

第五部分　引擎形态与部署选择

18　分离式推理架构

为什么拆分 Prefill 与 Decode、KV Cache 怎么传、代表性系统与全局动态路由。

18.1 **核心思想**

18.2 **为什么分离**

18.3 **KV Cache 传输**

18.3.1 PD 分离 KV Cache 传输模拟　代码

18.4 **代表性系统**

18.5 **PD 物理节点异步非对称编排**

18.5.1 P:D 配比怎么算

18.6 **KV Cache 跨节点高速传输协议**

18.6.1 定制化 RDMA 协议的关键优化

18.7 **全局动态路由**

18.7.1 Mooncake 的 KV Cache 为中心路由

18.8 **两个引擎里的分离实现**

18.8.1 vLLM 0.26.0：一对实例加一个连接器

18.8.2 SGLang 0.5.16：分离模式与 D 侧的预分配队列

18.8.3 生产上分离反而更慢的几种典型写法

章末产出　脚本 8　报告 8　结构化数据 11　日志 21

项目 18.1　**vLLM PD 分离连接器现役形态与 KV 传输代价模型**　**方案设计** 读透即可　案例解读

梳理 vLLM 现役 16 个 KV 连接器的注册名与收发角色写法，按模型配置手算每 token 的 KV 体积（Qwen2.5-7B 为 56 KiB），并给出用 /metrics 快照差量化 KV 传输的测量口径

交付可用连接器名单、KV 体积核算与 /metrics 传输测量口径；Qwen2.5-7B 每 token 56 KiB，PD 之间要搬多少字节由它乘 Prefill 长度直接算出。

参考环境未安装 nixl，NixlConnector 不可用、1P1D 未能起服，不含真 PD 的实测传输延迟与字节数。

项目 18.2 **多卡拓扑同口径压测与 PD 选型判定表** **实操** 读透即可

把合并单卡、合并 TP=2、双实例轮询压在同一套压测口径下，吞吐折成每卡再比，产出以每卡吞吐与 TTFT／ITL 高分位为落点的选型判定表，并写明四类反转条件

已实测合并单卡 914、合并 TP=2 450、双实例轮询 458 tok/s，均为每卡输出吞吐；按每卡算，切 TP 与起双实例都拿不回单卡的水平。

参考环境没有 InfiniBand / RoCE 网卡，真 PD 那一组未跑通、选型对照中 PD 一列无数据；双实例轮询组的 ITL 因代理是非流式口径而作废，不进选型。

19 主流推理框架对比

vLLM 与 SGLang 的关键参数详解，以及 Nano-vLLM 源码从入口开始的剖析；同一批请求两次跑出的 token 序列对不上，多半来自数值平局与批次组成扰动，不是调度器破坏了状态。

19.1 vLLM 关键参数详解

19.1.1 调度与批处理参数

19.1.2 KV Cache 与显存管理参数

19.1.3 量化与精度参数

19.1.4 并行参数

19.1.5 采样与生成参数

19.1.6 典型场景参数配置示例

19.1.7 参数联动与调优原则

19.1.8 vLLM 配置生成与显存预算 代码

19.2 Nano-vLLM 源码解读：从 example.py 入口剖析轻量级推理引擎

19.2.1 入口与配置：example.py

19.2.2 Nano-vLLM 整体架构：从 llm.generate() 到输出 token，数据流经六个组件

19.2.3 引擎初始化：LLMEngine

19.2.4 调度器：Continuous Batching + Chunked Prefill

19.2.5 块管理器：PagedAttention + Prefix Caching

19.2.6 ModelRunner：GPU 执行与 CUDA Graphs

19.2.7 输入准备：Prefill 与 Decode 的数据组装

19.2.8 模型前向：Qwen3 + TP 切分 + Flash Attention

19.2.9 采样：温度采样 + Gumbel-max 技巧

19.2.10 后处理与循环回写

19.2.11 一次 generate() 的完整时序

19.2.12 技术回链总表：19 处代码逐处对回前文章节

19.2.13 约 1200 行的 Nano-vLLM 覆盖调度、显存、执行、算子四层

19.3 SGLang 关键参数详解

19.3.1 调度与批处理参数

19.3.2 KV Cache 与显存管理参数

19.3.3 RadixAttention 与前缀缓存参数
19.3.4 量化与精度参数
19.3.5 并行参数
19.3.6 性能优化参数
19.3.7 结构化生成参数
19.3.8 典型场景参数配置示例
19.3.9 参数联动与调优原则
19.3.10 vLLM 与 SGLang 参数对照速查

章末产出 脚本 7 报告 8 结构化数据 47 日志 24

项目 19.1 **vLLM 参数调优：从 SLA 目标反推上线配置** **实操** 建议跑

网格扫 max_num_batched_tokens 与 max_num_seqs，把 SLA 翻译成三个可测字段；实测十二格全部超出 800 ms 首包门槛，给出扩容还是降负载的判定口径

网格十二格实跑，TTFT P99 全部超过 800 ms 门槛；交付网格口径、选型脚本与“当前 SLA 下没有可上线配置”这一结论。

三格为两个旋钮同时放大、无单旋钮对照；按峰值 QPS 限速的验证轮与改动后的复测未覆盖。

项目 19.3 **在 Nano-vLLM 上实现优先级调度** **从零实现** 必跑（2x4090）

在 nano-vllm 源码上把等待队列换成优先级堆并加抗饿死降权分支，实测交互类完成 P99 从 24.8 秒降到 8.6 秒、吞吐反升 65%，并用三条硬判据验证改造的正确性

三组实测完成，token 序列一致性 87/128，差异来自数值平局与批次组成扰动，不是调度器破坏了状态。

参考环境为 80G 单卡，三组抢占次数均为 0、抗饿死降权分支未被真正触发，第三组与优先级组同序、收益不计入，显存更紧或并发更大才会触发抢占；模型因下载超时由 Qwen3-0.6B 换为同架构的 Qwen3-8B。

20 Graph Compiler 与模型部署

torch.compile 与 TorchInductor、ONNX Runtime、TensorRT、TVM 的机制与对比选型。

20.1 **为什么需要 Graph Compiler**
20.2 **Torch Compile**
20.2.1 TorchDynamo
20.2.2 AOTAutograd
20.2.3 TorchInductor（代码生成后端）
20.2.4 torch.compile 推理加速 代码
20.2.5 推理引擎里的 torch.compile：为什么是分段编译
20.3 **TorchInductor 后端优化**
20.3.1 Kernel Fusion（算子融合）
20.3.2 Triton Backend（Triton 后端）

20.4 **ONNX Runtime**

20.4.1 ONNX Export（模型导出）

20.4.2 ONNX Graph Optimization（图优化）

20.4.3 PyTorch 导出 ONNX 代码

20.5 **TensorRT**

20.5.1 Builder

20.5.2 Engine

20.5.3 Runtime

20.5.4 Optimization Profile

20.5.5 TensorRT Engine 构建 代码

20.6 **TVM**

20.7 **AITemplate**

20.8 **Graph Compiler 对比**

20.9 **在锁定版本引擎里怎么用**

20.9.1 vLLM 0.26.0

20.9.2 SGLang 0.5.16

20.9.3 上线前最常见的几类故障与排查顺序

章末产出 脚本 5 报告 8 结构化数据 10 日志 9

项目 20.1 **torch.compile 推理加速实测** **实操** 建议跑

eager、default、reduce-overhead 三种模式同口径实测加速比与编译耗时：最快一档 1.29×，但编译要 97 秒、约 2.4 万次前向才能摊平编译开销，同时用重编译嫌疑标记暴露多形状风险

eager / default / reduce-overhead 三种模式乘三个形状实测：1×512 上 reduce-overhead 最快 1.29×，编译 97 秒、约 2.4 万次前向摊平编译开销。

max-autotune 一档与十个长度的动态形状扫表未覆盖；加速来自算子融合还是 CUDA Graph，尚无 kernel 层面的证据。

项目 20.3 **引擎选型评分卡与 TensorRT-LLM 构建路径梳理** **实操** 读透即可

在 TensorRT-LLM 装不上的真实约束下，用 vLLM 的 compile、enforce-eager、fp8 三组加 SGLang 对照组做一次只动一个因子的正交对比，产出一张缺实测就拒绝出总分的选型评分卡

vLLM 三组（compile 1627、enforce-eager 1549、fp8 2869 tok/s）与 SGLang 对照组实测完成：fp8 约为 compile 的 1.76 倍，而 compile 相对 enforce-eager 只快 5%。

TensorRT-LLM 在参考环境因 torch / CUDA 栈冲突无法安装，Engine 构建与 INT8 校准未执行，这两节交付命令模板与版本考据；FP8 GEMM 是否实际生效无 kernel 层面证据，量化后的质量评测未落盘。

21　新兴架构推理与长文本外推技术

两个引擎里的 Dynamic NTK 都是启动时算一次的静态方案，逐序列重算的只有参考实现；本章讲 RoPE 长度外推（YaRN / Dynamic NTK）与线性注意力、状态空间模型推理。

21.1　旋转位置编码 RoPE 外推（YaRN / Dynamic NTK）

- 21.1.1　RoPE 与长度外推的问题
- 21.1.2　Dynamic NTK
- 21.1.3　YaRN（Yet another RoPE extensioN）
- 21.1.4　RoPE 旋转位置编码与 YaRN 外推　代码
- 21.1.5　外推的代价：显存与每步时间

21.2　线性注意力与状态空间模型（Mamba / SSM）推理

- 21.2.1　线性注意力
- 21.2.2　状态空间模型（SSM）与 Mamba
- 21.2.3　Mamba-2 与混合架构

21.3　在锁定版本引擎里怎么用

- 21.3.1　长文本外推
- 21.3.2　Mamba 与混合架构
- 21.3.3　长上下文上线前的常见故障与排查顺序

章末产出　脚本 4　报告 10　结构化数据 9　日志 11

项目 21.1　**启用 YaRN 长度外推并验证效果**　**实操** 建议跑

按锁定版本修正 `rope_scaling` 配置启用 YaRN，用 5/5 passkey 大海捞针验证长上下文外推是否真有效

困惑度没有可用数据，结论只有 passkey 命中；factor=2 那一组只测到 16384 深度。

22　受控/结构化输出生成（Structured Decoding）

基于 FSM 的 JSON Schema 约束生成，以及 Outlines、XGrammar 等约束解码后端的选型。

22.1　基于 FSM 的 JSON 格式约束生成：Schema 约束过宽等于没有约束

- 22.1.1　自由生成的格式问题
- 22.1.2　基于 FSM 的约束方法
- 22.1.3　约束对生成质量的影响
- 22.1.4　FSM 约束的 JSON 结构化生成　代码

22.2 **开源生态与框架整合：约束解码的性能差异主要来自引擎实现，不是后端选型**

22.2.1 Outlines

22.2.2 Guidance

22.2.3 XGrammar

22.2.4 llguidance 与 lm-format-enforcer

22.2.5 后端选型

章末产出 脚本 11 报告 3 结构化数据 39 日志 29

项目 22.1 **用引擎原生结构化输出做 JSON 生成** **实操** 必跑（2x4090）

用 vLLM 原生结构化输出与 xgrammar 后端把 JSON Schema 合法率从 0% 提到 100%

有效样本 48 / 42 条，少于设计规模的 100 条；收益只到关约束与开约束两组。

项目 22.2 **约束解码后端的性能对比** **实操** 建议跑

串行同口径下只换约束后端：vLLM 上 xgrammar 与 outlines 差不到 5%，SGLang 上 outlines 每 token 耗时是 xgrammar 的 2.08 倍，差距可手算回掩码体积

三档串行墙钟实测完成，每档 40 条样本；结论只在串行口径下成立，高并发压测因客户端注不进 Schema 未跑。

参考环境的 vLLM 压测客户端没有 --extra-body，约束组注不进 Schema、走不了高并发压测；TTFT 全组未采；实测所用 SGLang 版本只暴露语法队列长度，语法缓存命中率无法验证。

项目 22.3 **约束对生成质量的影响评估与 Schema 编写规范** **实操** 建议跑

把一条从松到严的 Schema 阶梯放在同一把打分尺子下测合法率与字段级匹配率，分清是引擎拒收还是模型答错，产出五条能指回数据的 Schema 编写规范

质量侧实测：松档对业务 Schema 通过率 0%，业务档与修正后的严格档均为 100%（46 / 49 / 42 条有效样本），Schema 约束过宽等于没有约束。

原始严格档因 uniqueItems、multipleOf、idn-email 不在 xgrammar 支持子集内被 4xx 拒收，作为负例保留；延迟侧未跑，该章文档两张评估表的相应行交付命令与判定口径。

23 多模态大模型（MLLM）推理优化

视觉 Token 压缩后 TTFT 可能反而变差：实测把单图 token 由 2691 压到 1222，P99 TTFT 从 2799 升到 3190 ms。本章讲多模态推理管线、KV 前缀复用、图文混合负载的服务化，以及视频与多图把 Token 预算推高后的取舍。

23.1 **视觉 Token 压缩与 Prefill 优化**

23.1.1 视觉 Token 的开销分析

23.1.2 视觉 Token 压缩策略

23.1.3 Prefill 阶段优化

23.1.4 视觉 Token 压缩（Token Merging） 代码

23.2 **多模态推理管线：从一张图到第一个 Token**

23.2.1 管线解剖：预处理、编码、投影、生成

23.2.2 视觉编码器的计算画像
23.2.3 动态分辨率时代的 Token 预算
23.2.4 视觉 Token 计算器 代码
23.2.5 预处理这半截管线：CPU 上的隐形瓶颈
23.2.6 从像素到毫秒的首字预算器 代码

23.3 **多模态的 KV Cache 与前缀复用**

23.3.1 视觉 Token 的 KV 显存占用
23.3.2 同一张图只编码一次：内容哈希与前缀缓存
23.3.3 两级缓存的显存代价与临界命中率
23.3.4 多模态 KV 显存与并发计算器 代码

23.4 **图文混合负载的服务化**

23.4.1 长短 Prefill 互相堵死
23.4.2 部署要点与 TTFT 口径
23.4.3 三阶段分离：E/P/D 的趋势
23.4.4 多模态服务的监控清单

23.5 **视频与多图：Token 预算按乘法增长**

23.5.1 视频输入的 Token 数学
23.5.2 压缩与取舍的判断框架

章末产出 脚本 3 报告 2 结构化数据 5 日志 19

项目 23.2 **优化多模态模型的 Prefill 阶段** **实操** 建议跑

先算清一张图值多少视觉 token，再测端到端 TTFT 基线，只改 max_pixels 一处同口径复测；视觉 token 减半而首包 P99 反而变差，按预先登记的门槛判为不采纳

两组各 16/16 完成：单图视觉 token 由 2691 压到 1222，P99 TTFT 却由 2799 升至 3190 ms，按预先登记的门槛判为不采纳。

参考实测改用 Qwen2.5-VL-32B，课程原定 7B、权重下载超时；处理器侧那条基线未单独测数。

项目 23.3 **图文混合推理服务与端到端优化** **实操** 读透即可
案例解读

搭一套图文混合推理服务，跑出混合流量下的 TTFT 与吞吐基线，并给出长短 Prefill 互相挤占时的排查与定位方法

混合流量基线实测跑通：66.12 tok/s、P99 TTFT 2962 ms，12/12 完成；收视觉 token 上限的那一组未启动成功，本项目只给基线与排查方法，不给优化后的数。

混合流量下纯文本被挤压的幅度无实测。

24　推理系统性能度量与基准评测

TTFT / TPOT / ITL / Throughput 的定义与调优，压测工具、命令、报告解读与生产监控；均值改善而 P99 一起劣化，是压测里最常被误读的一种结果。

24.1　**TTFT、TPOT、ITL、Throughput 的定义与调优策略**

- 24.1.1　四个核心指标：TTFT、TPOT、ITL 与吞吐分别落在请求时间线的哪一段
- 24.1.2　指标间的权衡关系
- 24.1.3　四类优化目标的调优手段与关键参数对照

24.2　**经典压测与 Benchmarking 工具实践：不预热的第一轮会被 JIT 编译污染**

- 24.2.1　压测方法论：四类关键参数明确设定才可复现，不预热会被 JIT 编译污染统计
- 24.2.2　Benchmarking 工具
- 24.2.3　延迟-吞吐曲线分析

24.3　**压测命令实战（vLLM / SGLang 真实命令示例）**

- 24.3.1　vLLM 在线服务压测命令
- 24.3.2　vLLM 离线吞吐压测命令
- 24.3.3　SGLang 跨框架对比压测
- 24.3.4　三类典型场景的压测配置模板

24.4　**报告解读与瓶颈定位：均值改善而 P99 劣化，是尾延迟被均值掩盖**

- 24.4.1　压测报告的关键字段解读
- 24.4.2　从指标分布形态定位瓶颈
- 24.4.3　瓶颈定位决策树

24.5　**端到端调优案例（TTFT 过高→排查→验证）**

- 24.5.1　初始压测：发现 TTFT 超标
- 24.5.2　逐步排查与优化
- 24.5.3　优化后压测验证

24.6　**生产监控与告警**

- 24.6.1　关键监控指标体系
- 24.6.2　Prometheus + Grafana 看板配置
- 24.6.3　SLO 告警阈值设置

24.7　**LLM 推理服务压测脚本**　　代码

章末产出　脚本 5　　报告 4　　结构化数据 19　　日志 21

项目 24.1 **压测报告模板** **实操** 建议跑

用 vllm bench serve 做多轮重复压测，把多轮结果压成带 95% 置信半宽的一个数，并预先登记三条判定口径，产出可归档、可跨版本回归的压测报告模板

三轮实测：输出吞吐 1987±53 tok/s、P99 TTFT 253±9 ms；±53 就是这台机器的噪声带，小于它的差别不算差别。

官方的多轮 sweep 入口在实测机上会挂起，多轮改由手工三轮替代；没有改一处、复测一次的对照组，达标吞吐只在报告模板里有列、无实测值。

项目 24.2 **跨框架标准化对比与选型报告** **从零实现** 建议跑

先把参数对齐规则做成会拒绝出命令的校验器，再做 vLLM vs SGLang 的对比与选型

并发扫描与帕累托前沿未跑，对比只在并发 16、64 条一档、每边三轮。

项目 24.3 **TTFT 超标排查复盘** **实操** 建议跑

按预先登记的达标吞吐验收口径做一次 TTFT 改参实验：把单步 token 预算从 8192 收到 2048，用数据把一次方向看似正确的改参判成不通过并回滚

两组实测：单步 token 预算 8192 → 2048，P99 TTFT 792.20 → 944.54 ms、达标吞吐 27.07 → 25.18 req/s；平均 TTFT 变好而高分位与达标吞吐一起变差，假设被证伪、未过验收。

基线只压到贴着 800 ms 门槛，不是该章文档设定的那种已经 2 秒的事故；指标快照缺排队与 prefill 两个直方图，排队与 prefill 的分段归因未覆盖。

25 Profiling与Runtime调度

先测准再调度：nsys / ncu / torch.profiler 三层怎么分工、GPU 性能分析指标怎么读，以及主流框架 Runtime 与优化实践；A/B 两组只要 KV 池没对齐，吞吐差里就混着第二个变量。

25.1 Profiling 基础

- 25.1.1 Profiling 的目标
- 25.1.2 GPU 推理流水线分析
- 25.1.3 七类瓶颈：Decode 在 Kernel 发射与带宽，Prefill 在算力
- 25.1.4 常用 Profiling 工具
- 25.1.5 Profiling 实战流程
- 25.1.6 Profiling 工具的安装

25.2 CUDA Profiling

- 25.2.1 CUDA Event 测时
- 25.2.2 NVTX Range 标记
- 25.2.3 Nsight Systems 时间线分析
- 25.2.4 CUDA Kernel Profiling 代码

25.3 PyTorch Profiling

- 25.3.1 torch.profiler 基本使用
- 25.3.2 Trace 分析

25.3.3 Memory Profiling
25.3.4 算子热点定位
25.3.5 PyTorch Profiler 性能分析 代码

25.4 **GPU 性能分析指标**

25.4.1 Memory Bandwidth
25.4.2 Roofline Model
25.4.3 Compute Bound 与 Memory Bound
25.4.4 Occupancy 与 Warp Stall 分析

25.5 **Runtime 基础**

25.5.1 模型加载的三个关键工程细节：safetensors 零拷贝内存映射、逐层加载加异步 H2D、权重预热
25.5.2 图执行
25.5.3 CUDA Graph
25.5.4 CUDA Graph 推理优化 代码

25.6 **主流推理框架 Runtime**

25.7 **Runtime 性能优化实践**

25.7.1 Runtime 调优手段速查
25.7.2 构建一个 Mini LLM Runtime 代码

25.8 **先测准再调度：nsys / ncu / torch.profiler 三层定位与框架选型**

章末产出 脚本 9 报告 5 结构化数据 22 日志 36 profile 产物 6

项目 25.1 **用 torch.profiler 定位推理热点算子** **实操** 必跑（2x4090）

用 record_function 把推理切成 prefill 与 decode 两段做 torch.profiler 采集，读算子排行与 GPU 占空比，再换 sdpa 复测；实测本微基准上 sdpa 反而更慢

整段算子表与端到端墙钟实测：eager 1.008 s vs sdpa 1.331 s，换 sdpa 未过判定。

prefill / decode 的分段耗时未采集，本项目不给分阶段结论。

项目 25.2 **对 vLLM / SGLang 服务做 Nsight 端到端 Profiling** **实操** 建议跑

为常驻 vLLM / SGLang 服务设定采集窗口做 nsys 端到端 Profiling 定位开销最高的 kernel，再用 ncu 下钻看显存与算力吞吐，按预先登记的判定标准决定改动采不采纳

ncu 实测两引擎主 GEMM 均为访存受限（显存吞吐约 86%、算力约 12%）；单步 token 预算翻倍在两侧均被否决：vLLM TTFT P99 839 → 886 ms、SGLang 801 → 1250 ms。

ncu 只落了显存与算力两项吞吐；复测组沿用基线的微观读数。

项目 25.3 **复现 CUDA Graph 捕获机制并量化 Decode 加速收益** **从零实现** 建议跑

读懂两引擎的捕获与回放实现，在最小 Runtime 里自行复现捕获与回放并主动触发一次捕获失败，再以 enforce-eager 为对照组做引擎侧 A/B，量化 decode 加速

引擎侧 A/B 实测 2522 vs 1320 tok/s，手写 Runtime 四档加速比 1.16–1.18×；1.91× 那组含第二变量（禁用 CUDA Graph 后 KV 池变大），不可读作算力提速。

捕获显存与服务冷启动耗时未单独测。

26 推理系统工程化与生产部署

容器化与 Kubernetes 部署、Ray Serve、高可用、多租户隔离、扩缩容与成本优化；GPU 节点设置了污点、Pod 不配置容忍度就无法调度到该节点，按 CPU 利用率扩容则反映不了 GPU 负载。

26.1 部署架构：离线批量同样用连续批处理，批次上限调到远高于在线

- 26.1.1 在线服务（Online Serving）
- 26.1.2 离线批量（Offline Batch）
- 26.1.3 API 网关（API Gateway）
- 26.1.4 三种部署模式对比
- 26.1.5 部署选型：单机 Docker Compose 起步，生产级 K8s + HPA

26.2 Docker 容器化：LLM 镜像体积大、依赖驱动、权重外置挂载，要专门优化

- 26.2.1 Docker 核心概念速查
- 26.2.2 Dockerfile 编写
- 26.2.3 GPU Runtime
- 26.2.4 镜像优化
- 26.2.5 vLLM Docker 部署 代码

26.3 Kubernetes 编排：GPU 节点设置了污点，Pod 不配置容忍度就无法调度到该节点

- 26.3.1 K8s 核心概念速查
- 26.3.2 GPU Operator
- 26.3.3 Device Plugin
- 26.3.4 Deployment
- 26.3.5 HPA
- 26.3.6 Helm
- 26.3.7 vLLM Kubernetes 部署 代码

26.4 Ray Serve：只有超出单机或需要多模型 DAG 时才考虑

- 26.4.1 Ray Actor
- 26.4.2 Autoscaling
- 26.4.3 Replica
- 26.4.4 Ray Serve 与 K8s 方案的选型

26.5 **高可用架构：模型加载以分钟计，maxUnavailable 必须设为 0**

26.5.1 滚动升级

26.5.2 金丝雀发布

26.5.3 蓝绿部署

26.5.4 优雅退出

26.6 **多租户隔离：MIG、MPS、时间片的隔离强度依次递减，时间片连显存都不隔离**

26.7 **三层扩缩容：按 QPS 与 KV 占用扩，CPU 利用率反映不了负载**

26.8 **成本优化：Spot 实例是可压缩成本里占比最大的一项，代价是必须可重试且多副本跨可用区**

26.8.1 GPU 利用率优化

26.8.2 Spot 实例

26.8.3 PD 分离

26.8.4 Prefix Cache 与 KV 复用

26.8.5 批处理合并

26.9 **可观测性、告警与容量规划：冗余度由 SLA 决定**

26.9.1 生产级 LLM Serving 集群部署 代码

章末产出 脚本 1 报告 3 结构化数据 3 日志 4

项目 26.1 **vLLM 服务容器化与镜像 / 启动时间优化方案**

实操 读透即可
案例解读

做镜像分层体积审计，给出多阶段瘦身与编译缓存持久化的优化方案；本地从启动到服务就绪的墙钟实测 54 秒

本地冷启动 54 秒已实测；交付 Dockerfile、分层体积审计脚本与瘦身方案，容器内的启动时间未测，54 秒是裸机口径。

实测机未安装 Docker，镜像体积、层数、容器内冷热启动对比与编译缓存对照均无实测数据。

项目 26.2 **Kubernetes + HPA 部署方案设计与扩缩容验收口径**

方案设计 读透即可
案例解读

用 GPU Operator 与 Device Plugin 编写 Kubernetes 部署编排与自定义指标 HPA 配置，给出扩容延迟的四段构成与 SLA 违约窗口的测算口径和验收表

交付 K8s 部署与 HPA 配置文件、压测与验收方案；HPA 指标要选 vllm:num_requests_waiting，选 num_requests_running 会被 max-num-seqs 顶住天花板、永远触发不了扩容。

实测机没有 Kubernetes 环境，扩缩容全程未在真机跑过，不含实测扩缩容数据。

项目 26.3 **高可用推理集群方案**

方案设计 读透即可
案例解读

产出含 Kubernetes 高可用、多租户隔离三档、告警最小集与容量成本模型的集群设计文档；成本按满足 SLA 的实测饱和 QPS 反推，缺实测值时脚本拒绝出数

交付高可用与故障注入的设计文档与配置文件；成本模型中单卡满足 SLA 的饱和点 18.86 QPS 引自第 1 章实测，峰值 40 QPS 为场景设定。

与上一项同为设计交付：实测机没有 Kubernetes 环境，高可用与故障注入未在真机验证过；本课不提供任何市场价格数字。